프로그람개발문고

KDE/Qt프로그람작성법

교육위원회 교육정보쎈터

주체99(2010)

차　례

머리말

위대한 령도자 **김정일**동지께서는 다음과 같이 지적하시였다.

《정보산업을 빨리 발전시키고 인민경제의 모든 부문을 정보화하여야 합니다.》

(《**김정일**선집》 제15권, 114페지)

위대한 령도자 **김정일**동지의 현명한 령도에 의하여 오늘 우리 나라에서는 인민경제의 정보화를 실현하기 위한 투쟁이 힘있게 벌어지고있다.

인민경제의 정보화를 실현하고 정보산업을 하루빨리 세계적수준에 끌어올리기 위해서는 정보기술 특히 프로그람기술을 빨리 발전시키고 능률적인 프로그람들을 적극 개발하여야 한다.

이 책에서는 《붉은별》과 Linux조작체계에서 실행할수 있는 프로그람들을 개발하기 위한 개발도구인 Qt와 KDE에 기초한 GUI프로그람작성법에 대하여 설명한다.

1장에서는 Qt와 KDE서고를 가지고 대체로 수행하는 일과 각종 클라스들을 사용하여 실행프로그람을 구축하는 방법에 대한 기초지식을 학습한다.

2장에서는 Qt와 KDE응용프로그람들에서 기본창문을 만들고 현시하는데 사용하는 클라스들을 설명한다.

3장에서는 창문과 대화칸들의 내용을 조직하는 방법을 설명한다.

4장과 5장에서는 튀여나오기대화칸, 사용자정의대화칸, 그리고 Qt와 KDE의 부분으로 정의된 대화칸들에 대하여 학습한다.

6장에서는 차림표와 도구띠의 구축과 관리방법을 설명한다.

7장에서는 on과 off상태를 절환하는 단추집합과 같이 그룹에 포함되는 창문부품들의 관리방법을 설명한다.

8장에서는 프로그람이 마우스와 단추에 응답하는 방법을 설명한다.

9장에서는 파일로부터 적재되거나 프로그람으로 콤파일된 화소준위도형을 관리하고 현시하는 방법을 설명한다.

10장에서는 서체와 문자렬현시과정을 설명한다.

11장에서는 색을 만들고 관리하는데 사용하는 옵션들을 설명한다.

12장에서는 QPainter클라스를 리용하여 넓은 범위의 자세한 도형그리기를 수행하는 실례들을 설명한다.

13장에서는 특정크기에 맞게 도형을 조작하거나 임의의 페지에 배치하는 과정 그리고 동화상에 대하여 설명한다.

14장에서는 도형 또는 본문객체를 끌어다놓는 실례들을 설명한다.

15장에서는 응용프로그람들사이의 자료를 통신하는 과정을 설명한다.

16장에서는 기타 편의클라스들의 실례를 설명한다.

17장에서는 국제화를 위한 KDE편의클라스들을 설명한다.

18장에서는 Qt창문부품들을 자모순으로 설명한다.

19장에서는 KDE창문부품들을 실례를 들어 설명한다.

20장에서는 Win32, KDE, GNOME으로 작성한 같은 프로그람들의 실례를 설명한다.

제1장. KDE의 기초

학습내용

KDE의 전체 구조와 매개 구성요소들을 리해

Qt가 프로그람개발에서 차지하는 역할

KDE가 프로그람개발에서 차지하는 역할

창문부품과 사건모형

쏘프트웨어의 이름은 K Desktop Environment 간단히 KDE라고 부른다. 이 장은 KDE의 응용프로그람개발환경에 대한 소개이다. KDE는 Linux와 UNIX계렬조작체계에서 인기있는 도형방식사용자대면부이다. 사실상 UNIX계렬의 모든 도형방식대면부들은 X창문체계상에 구축된다. X창문체계는 많은 체계들에서 이식가능한 도형방식을 주며 도형객체들의 Qt서고는 응용프로그람의 기본구축블로크를 제공하며 KDE서고는 표준형식을 제공한다.

제1절. KDE응용프로그람의 구조

KDE응용프로그람은 다른 많은 코드에 기초하여 작성한다.

응용프로그람을 작성하는 구체적인 일은 대부분 이미 수행되여있으며 그것은 수행하려는 응용프로그람에 련결되는 코드서고에 상주한다.

그림 1-1은 KDE응용프로그람을 만드는 쏘프트웨어준위들에 대한 개념을 준다.

응용프로그람	
KDE 클라스	
Qt 클라스	
C++ API	
glib	X11
조작체계	

그림1-1. Linux에서 KDE응용프로그람에 대한 쏘프트웨어준위들

그림에서는 준위들이 완전히 독립되여있지만 실제로 그렇지 않다. 례를 들면 실제의 호출은 KDE클라스로부터 glib함수들에로 이루어지며 응용프로그람은 직접호출 즉 glib나 체계호출을 방해하지 않는다. 응용프로그람은 일반적으로 KDE와 Qt로부터 클라스들을 사용한다. 그러나 호출은 오직 아래로 향한다. 실례로 Qt API의 부분은 KDE에서 어떤것도 사용하지 않는다.

이 책은 2준위와 3준위(KDE와 Qt)의 기능을 리용하여 제일 웃준위에서 동작하는응용프로그람을 만드는 방법을 주며 다른 준위의 사용법에 대하여서는 거의 설명하지 않는다. 실제로 그것은 KDE와 Qt의 중요한 목적중의 하나로서 프로그람작성자를 낮은 준위의 구체적인 조종으로부터 격리시킴으로써 응용프로그람개발과정을 간단화한다.

제2절. 쏘프트웨어준위

이 절에서는 그림 1-1에서 보여주는 쏘프트웨어준위들을 설명한다.

1. 체계

체계준위는 모든 Linux응용프로그람에서 사용할수 있는 쏘프트웨어의 제일 아래층이다. 일부 아래준위의 체계호출은 조작체계와 그 구동프로그람들에로의 직접호출을 제공하여 파일열기와 등록부창조와 같은 일을 수행한다. Linux핵심은 C로 씌여지므로 이것들은 모두 C함수호출이다.

2. glib

glib는 우의 모든 층들에서 사용되는 C함수, 마크로 그리고 구조체들의 모임이며 응용프로그람에서도 사용한다. glib서고는 기억할당, 문자렬형식화, 날자와 시간, 입출력, 그리고 시계용 함수를 포함한다. 또한 련결목록, 배렬, 하쉬표, 나무, 쿼크 및 캐쉬용 편의함수들을 가지고있다. glib에 의해 조종되는 중요한 함수들중 하나는 기본순환고리로서 KDE가 응용프로그람의 코드들을 동시에 실행하는 여러 자원을 조종하게 한다.

3. X11

이것은 현시를 조종하는데 사용되는 저준위함수들을 포함하는 도형처리층이다. 모든 기본창문함수들이 포함되며 이 함수들은 창문을 현시하고 마우스와 건반사건에 응답한다. 이 서고는 여러해동안에 매우 안정되였으며 판번호는 그리 변화하지 않았다. 현재 그 판번호는 11(이름이 가리키는것처럼)이다. 그리고 판 11이 출하물 6에 있으므로 X11R6으로서 알려져있다. 그 원시이름에는 판번호가 없었으므로 흔히 간단히 X라고 불리운다.

4. C++ API

이 층은 모두 C++로 씌여졌으므로 C++실행시체계는 새 객체창조와 I/O흐름조종과 같은 것을 위해 호출된다.

5. Qt클라스

이 층의 C++모임은 응용프로그람을 만드는데 사용되는 여러가지 창문부품들(단추, 창틀 등)을 실현하며 창문들과 결합하여 복잡한 도형방식대화칸들을 만드는 능력을 가지고있다. 창문부품들을 현시하는것과 함께 입력시에 마우스와 건반사건에 응답할수 있고 입력창

문으로부터 필요한 프로그람부분에로 정보를 발송한다.

6. KDE클라스

이 클라스들은 Qt클라스들을 수정하고 기능을 추가한다. 많은 KDE클라스가 있으나 그 대부분은 하나이상의 Qt클라스들로부터 직접 확장된다. 이 층은 KDE에 그 유일한 외관을 주고 창문, 마우스 그리고 건반 모두가 서로 교제하는 방법을 표준화한다.

7. 응용프로그람

응용프로그람의 2가지 기본특성은 Qt응용프로그람 그리고 KDE응용프로그람을 창조할수 있는것이다. Qt응용프로그람은 QApplication객체를 만들고 초기화하며 KDE응용프로그람은 KApplication객체를 만들고 초기화한다. KApplication클라스는 KDE응용프로그람의 표준외관과 필요한 기능들을 추가함으로써 QApplication클라스를 확장한다.

제3절. Qt에 대하여

Qt는 C++ GUI응용프로그람개발쏘프트웨어의 서고이다. Qt의 목적은 응용프로그람의 사용자대면부를 개발하는데 필요한 모든 기능을 제공하는것으로서 주로 C++클라스모임의 형식으로 실현된다.

노르웨이의 TrollTech회사(http://www.trolltech.com)는 1995년에 처음으로 상업용 Qt를 소개하였다.

Qt클라스모임은 매우 든든하며 Qt를 사용하여 완전한 응용프로그람을 작성할수 있다. 사실상 응용프로그람의 기본형식을 보여주기 위하여 이 책에서 처음의 몇가지 실례는 오직 Qt만 사용한다. Qt클라스는 기본창문조종, 끌어다놓기 그리고 망프로그람작성을 위한 국제화에 이르기까지의 모든 기능을 포함한다.

이전에 쏘프트웨어사용허가제한들로 인하여 공개원천개발분야에서는 Qt의 사용에서 일정한 제한이 있었다. 그러나 사용허가는 더이상 필요없게 되였다. 최근에 Trolltech는 완전히 자유로운 Qt판을 배포하였으며 그것은 GPL(GNU General Public License)하에서 허가된다. 또한 같은 쏘프트웨어를 QPL(Q Public License)하에서도 사용할수 있다. 이러한 2중사용허가는 공개원천쏘프트웨어와 소유권있는 쏘프트웨어개발을 모두 허용한다.

Qt 2.2.1배포물에는 쏘프트웨어의 3개의 판이 있다.

· Qt Free Edition은 GPL하에서 허가되고 내리적재할수 있고 어떠한 공개원천프로젝트에도 자유롭게 사용할수 있다.

· Qt Professional Edition은 상업적이고 소유권있는 쏘프트웨어개발에서 사용된다. 사용허가와 쏘프트웨어를 구입하여야 한다.

·Qt Enterprise Edition은 Qt Professional Edition과 같이 허가되지만 추가적인 쏘프트웨어모듈을 포함한다. 이 확장은 OpenGL, 망기술, XML, 표계산 그리고 특별히 최적화된 2D도형처리묶음을 포함한다.

1. QObject클라스

대부분의 QObject클라스들은 기초클라스 QObject를 계승한다. 이것은 사실상 Qt서고의 모든 클라스들이 꼭같은 메쏘드들의 기본모임을 포함한다는것을 의미한다. QObject클라스의 구성자는 마음대로 부모객체의 주소와 이름을 객체에 할당하는 문자렬을 받아들일수 있다.

```
QObject ( QObject *parent = 0 , const char *name = 0 ) ;
```

다음 메쏘드들은 QObject에 선언되여있다.

```
void blockSignals(bool b);
QObject *child(const char *name, const char *type = 0);
const QObjectList *children() const;
virtual const char *className() const;
static bool connect(const QObject *sender, const char *signal,  const QObject *receiver, const char *member);
bool connect(const QObject *sender, const char *signal, const char *member) const;
static bool disconnect(const QObject *sender,
const char *signal, const QObject *receiver, const char *member);
bool disconnect(const char *signal = 0, const QObject *receiver = 0, const char *member = 0);
bool disconnect(const QObject *receiver, const char *member = 0);
void dumpObjectInfo();
void dumpObjectTree();
virtual bool event(QEvent *);
virtual bool eventFilter(QObject *, QEvent *);
bool highPriority() const;
bool inherits(const char *) const;
virtual void insertChild(QObject *);
void installEventFilter(const QObject *);
bool isA(const char *) const;
bool isWidgetType() const;
void killTimer(int id);
void killTimers();
virtual QMetaObject *metaObject() const;
const char *name() const;
const char *name(const char *defaultName) const;
static const QObjectList *objectTrees();
QObject *parent() const;
QVariant property(const char *name) const;
QObjectList *queryList(const char *inheritsClass = 0, const char *objName = 0,  bool regexpMatch = TRUE,
    bool recursiveSearch = TRUE);
```

```
virtual void removeChild(QObject *);
void removeEventFilter(const QObject *);
virtual void setName(const char *name);
bool setProperty(const char *name, const QVariant &value);
bool signalsBlocked() const;
int startTimer(int interval);
QStringList superClasses(bool includeThis = FALSE) const;
static QString tr(const char *);
```

이 메쏘드의 대부분은 이 책의 여러 실례들에서 사용된다. 일부 Qt객체들은 프로그람안의 다른 객체가 받아들일수 있는 신호를 발생시킬수 있는 능력을 가지고있다. QObject객체는 해체자가 호출될 때마다 다음 신호를 발생시킨다.

```
void destroyed();
```

신호와 그것을 받아들이는 처리부는 주로 다음 장에서 설명하며 이 책에는 많은 실례들이 포함되여있다.

2. MOC콤파일러

개발자가 사용하는 한가지 프로그람은 메타객체콤파일러(또한 MOC콤파일러)이다.

MOC콤파일러는 원천코드를 읽어들여 응용프로그람을 콤파일, 련결하는데 필요한 특별한 C++원천파일들을 생성한다. 이 특수파일들은 다른 객체들의 하나이상의 《처리부》들이 받아들이는 《신호》를 발생시키는데 필요한 코드를 포함한다. 신호는 응용프로그람안의 한 객체에서 다른 객체에로 정보를 비동기적으로 전송하는데 사용되는 메쏘드이다.

MOC콤파일러는 클라스선언에 Q_OBJECT마크로가 존재하는가에 따라서 코드를 생성하겠는가 생성하지 않겠는가, 어떤 코드를 생성하겠는가를 결정한다. 결과의 원천코드는 따로따로 콤파일되고 련결되거나 또는 간단히 #include지령에 의해 코드에 포함된다.

MOC콤파일러의 리용은 신호와 처리부들을 동작시킬뿐아니라 모든 Qt클라스에서 (그리고 계승에 의해 프로그람의 모든 객체에서) 선언된 특별한 메쏘드들을 리용할수 있는 코드를 생성한다. 표 1-1에 보여주는 이 메쏘드들은 QObject클라스에서 선언된다.

표 1-1. QObject의 MOC메쏘드들

메쏘드	설명
className ( )	클라스이름을 문자렬로 돌려준다. 이것은 RTTI(실행시형식별)기능을 요구하지 않는다.
inherits()	이 클라스가 다른 클라스를 계승하는가 아닌가를 지정하는 론리값을 돌려준다.
tr()	국제화를 위한 문자렬번역을 수행한다.
setProperty()	이름에 의해 객체속성을 설정한다.
property()	이름있는 객체속성을 돌려준다.

메쏘드	설명
metaObject()	클라스의 QMetaObject객체를 돌려준다. 메타객체는 클라스에 대한 자세한 설명정보를 포함한다.

제4절. KDE에 대하여

KDE는 도형방식탁상환경의 공개원천개발계획이다. 준말의 첫 문자 K는 의미를 가지지 않고 단지 이름이다.

KDE쏘프트웨어는 Qt를 사용하여 구축한다. 계획은 Qt의 첫판본이 공개된 1996년에 시작되였으며 그후 점차 응용프로그람의 큰 집합을 가진 매우 완전한 탁상환경으로 되였다.

쏘프트웨어개발자의 견지에서 KDE는 매우 간단하다. KDE계획의 일부로서 씌여진 쏘프트웨어의 대부분이 탁상환경의 완전한 부분으로 사용되고있으며 많은 클라스들이 개발되여 핵심 KDE API의 부분으로 포함되였다. 이 클라스들은 KDE응용프로그람에 표준형식을 제공한다. 클라스들의 대부분은 Qt서고의 하나이상의 클라스들을 계승하며 KDE클라스중 일부는 Qt의 클라스들에 기능을 추가하지만 그 대부분은 간단히 KDE의 표준외관을 유지관리하기 위한것이다. 오직 Qt의 클라스들만 리용하여 모든 응용프로그람을 작성하는것은 대단히 쉽지만 KDE클라스들을 리용하면 응용프로그람은 탁상에 의거하여 통합된것처럼 보인다.

제5절. 사건의 발생

K Desktop Environment에서 실행하는 응용프로그람은 사건구동형프로그람이다. 사건구동이라는것은 프로그람이 실행을 시작할 때 창문(또는 창문들)을 현시하고 마우스나 건반으로부터 입력을 기다린다는것을 의미한다. 입력은 사건이라고 하는 객체에 보관된다. 또한 사건은 프로그람에게 창문이 닫기였거나 창문이 다른 창문뒤에 가리워졌다가 로출되였다는것을 알릴수 있다. 응용프로그람의 목적은 건반과 마우스사건에 지능적으로 응답하는것이다.

응용프로그람은 1개의 기본적인 제일웃준위창문을 가지고있다. 또한 다른 창문들을 가질수 있다. 창문은 응용프로그람의 생명주기 전기간 존재하거나 응용프로그람이 사건에 응답할 때 나타나거나 사라질수 있다.

매개 창문은 창문부품에 은페된다. 응용프로그람의 제일웃준위창문은 창문부품이다. 매개 튀여나오기창문도 역시 창문부품이다. 사실상 현시기는 창문부품들로 구성된다. 하나의 창문부품은 다른 창문부품들을 포함하고 현시할 능력이 있으며 모든 단추, 표식자 그리고 차림표항목은 개별적인 창문부품이다. 응용프로그람의 도형방식대면부의 프로그람작성은 창문부품들을 만들고 결합한 다음에 창문부품들을 능동으로 하고 창문부품들이 받아들인 사건들에 응답하는 코드를 쓰는 문제이다.

창문부품은 QWidget라는 Qt클라스로부터 계승되는 클라스들이다. QWidget객체는 자체의

현시가능창문을 포함하고 관리한다. 또한 마우스와 건반에 의해 발생하여 창문부품내의 창문에 보내오는 사건들에 응답하도록 설정될수 있다. 사건들은 현재의 보임상태, 크기, 배경색, 전경색, 현재위치 등을 포함하며 Qt 또는 KDE에서 선언된 창문부품들을 사용하거나 QWidget를 기초클라스로 사용하여 창조할수 있다.

제6절. 객체의 이름

Qt클라스이름은 문자 Q로 시작되고 KDE클라스이름은 문자 K로 시작된다. 그러므로 프로그람의 원천코드를 읽을 때 클라스가 선언되는 서고를 결정할수 있다. 첫 문자외에 같은 이름을 가지는 2개 클라스가 있으면 그것은 한 클라스가 다른 클라스의 확장이라는것을 의미한다. 실례로 KDE클라스 KPixmap은 그 기초클라스로서 Qt클라스 QPixmap를 사용한다.

Qt와 KDE에서 모든 클라스들은 머리부파일에서 선언된다. 거의 모든 경우에 머리부파일의 이름은 클라스의 이름으로부터 알수 있다. 실례로 QPopupMenu클라스의 머리부파일은 qpopupmenu.h이고 클라스 KfontDialog는 kfontdialog.h에서 선언된다. 그러나 둘이상의 클라스가 하나의 머리부파일에서 선언될수 있으므로 명명관례는 일반적으로 옳지 않다. 실례로 클라스 KFontChooser는 kfontdialog.h에서 선언된다. 또한 일부 원천파일이름은 간략된다. 실례로 KColorDialog의 머리부는 kcolordlg.h이다.

요 약

이 장은 KDE의 프로그람작성환경에 대하여 아주 간단히 그리고 일반적으로 소개하였다. 이 장에서 소개한 개념들은 다음과 같다.

· 일부 쏘프트웨어층은 KDE쏘프트웨어서고를 유지한다.

· X창문체계는 아래준위GUI대면부를 조종한다. KDE서고는 그 바로 아래에 있으며 Qt쏘프트웨어에 중요하게 의존하는 쏘프트웨어층이다.

· 모든 응용프로그람은 사건구동형이다. 응용프로그람은 적어도 하나의 창문을 현시한 다음 마우스 또는 건반으로부터 입력을 기다린다.

(자기가 Windows프로그람작성자이지만 KDE에 익숙되지 못했다면 KDE프로그람과 Windows프로그람을 비교하는 20장을 읽으시오. 그렇지 않으면 2장으로 넘어가서 아주 간단한 KDE응용프로그람의 실례로부터 학습을 시작하시오.)

제2장. 창문의 창조와 현시

학습내용

몇행의 코드로 간단한 프로그람의 작성

KDE프로그람의 실현이 간단히 추가된 능력을 갖춘 Qt프로그람의 실현

C++객체를 코드화하여 창문을 창조하기

신호를 받아들일 처리부를 지정하여 입력에 응답하는 방법

쏘프트웨어의 이름은 K Desktop Environment 간단히 KDE라고 부른다. 이 장은 KDE이 장은 Qt 또는 KDE응용프로그람의 기본형식을 설명한다.

이 장의 실례들은 원천의 기본형식과 그 원천을 실행가능프로그람으로 바꾸는데 필요한 과정을 설명하도록 설계되였다. 프로그람의 여러 부분사이의 관계를 될수록 쉽게 리해하도록 하기 위하여 실례들은 모두 간단히 손으로 작성한 makefile들을 사용한다. 처음의 실례는 간단한 Qt응용프로그람이고 두번째는 KDE응용프로그람이다. 다른 실례들은 어떻게 누름단추사건에 응답하고 다른 창문부품들을 포함하는 현시창문부품을 만드는가를 보여준다.

제1절. Hello Qt

다음 실례프로그람은 간단한 창문을 만들어 표시한다. 그것은 본문행을 현시하는것외에 다른것은 없지만 Qt프로그람의 기본형식을 보여준다. 실행결과는 그림 2-1에 보여준다.

```
1 /* helloworld.cpp */
2 #include <qapplication.h>
3 #include <qlabel.h>
4 #include <qstring.h>
5
6 int main(int argc, char **argv)
7 {
8     QApplication app(argc, argv);
9     QLabel *label = new QLabel(NULL);
10    QString string("Hello, world");
11    label->setText(string);
12    label->setAlignment(
13            Qt::AlignVCenter | Qt::AlignHCenter);
14    label->setGeometry(0,0,180,75);
15    label->show();
16    app.setMainWidget(label);
17    return(app.exec());
18 }
```

그림 2-1. 본문을 현시하는 간단한 Qt프로그람

2행에서 포함한 파일 qapplication.h는 main()함수가 들어있는 원천파일을 가지고있다. 이 실례는 본문을 현시하는데 QLabel창문부품을 사용하므로 qlabel.h를 포함하여야 한다. 그리고 QLabel객체에 현시되는 본문을 지정하는데 QString객체를 요구하므로 qstring.h를 4행에서 포함한다.

8행은 app라고 부르는 QApplication객체를 만든다. QApplication객체는 응용프로그람의 제일웃준위창문(또는 창문의 모임)을 보유하는 용기이다. 응용프로그람의 제일웃준위창문은 절대로 부모창문을 가지지 않는다. QApplication객체가 객체들을 넘겨받고 응용프로그람을 관리하므로 이 객체는 프로그람마다 오직 하나만 있을수 있다. 또한 QApplication객체의 창조는 Qt체계를 초기화하므로 다른 Qt기능들을 사용하기전에 존재하여야 한다.

Qt프로그람은 C++프로그람이다. 이것은 프로그람을 기동할 때 main()이라는 함수가 조작체계에 의해 호출된다는것을 의미한다. 그리고 모든 C++프로그람들처럼 지령행선택들을 main()에 넘길수도 있고 넘기지 않을수도 있다. 지령행선택은 8행에서처럼 초기화과정의 부분으로서 Qt쏘프트웨어에 넘어간다.

2개의 지령행인수 argc와 argv는 특별한 기발과 환경설정을 지정할수 있으므로 app의 구축에 사용된다. 실례로 Qt프로그람을 –geometry로 기동하여 그것이 현시되는 창문의 크기와 위치를 지정한다. 사용자는 프로그람을 기동하는 속성(profile)정보를 바꿈으로써 프로그람의 모양을 개성화할수 있다.

QLabel창문부품은 9행에서 만들어진다. QLabel창문부품은 간단히 문자렬을 현시할수 있는 창문이다. 표식자는 제일웃준위창문이고 제일웃준위창문은 부모가 없으므로 그 부모창문부품이 NULL로서 만들어진다. 표식자는 창조될 때 본문을 가지지 않고 10행에서 만들어진 QString객체를 넘겨받는다.

QString객체는 11행에서 setText()호출에 따라 QLabel에 설정된다. QLabel의 기정동작은 문자렬을 수직으로 중심에, 왼쪽으로 맞추어 현시하는것이므로 12행에서 setAlignment()를 호출하여 본문을 수직뿐아니라 수평으로도 중심에 배치한다.

14행에서 setGeometry()호출은 QApplication창문안에 있는 표식자창문부품의 위치, 높이 그리고 너비를 결정한다. 이 실례에서 표식자는 기본창문의 왼쪽웃구석(0,0)에 놓인다. 또한 180화소너비와 75화소높이로 지정된다. 기본창문은 현시하기전에 표식자의 크기를 알아야 표식자를 포함하기 위한 창문의 크기를 설정할수 있다.

16행의 show()호출은 표식자를 창문에 실제로 표시한다. show()함수는 즉시 창문부품을

현시하지 않고 시간이 되였을 때에만 현시하도록 환경을 구성한다. 부모창문(이 경우에 QApplication창문)은 표식자를 현시하는 일은 표식자의 show()호출이 있어야만 가능하다. hide()라는 다른 함수는 현시된 창문부품이 은폐되게 한다.

11행의 setMainWidget()호출은 기본창문에 표식자를 삽입한다. 이 실례에서는 QLabel객체가 사용되지만 보통 창문부품은 창문부품, 본문 그밖에 응용프로그람의 기본창문의 각이한 요소들의 모임을 포함하는 합성창문부품이다. 끝으로 17행에서 exec()호출이 이루어진다. 이 함수는 프로그람이 실행을 멈출 때까지 귀환하지 않는다. exec()는 그 완성상태를 표시하는 int값을 돌려주며 우리가 그 상태코드를 처리하지 않으므로 값은 단순히 체계에로 되돌아간다.

프로그람이 간단하고 오직 하나의 원천파일로 이루어져 있으므로 그것을 콤파일하는 makefile도 아주 간단하다.

```
INCL= -I$(QTDIR)/include -I$(KDEDIR)/include
CFLAGS= -pipe -O2 -fno-strength-reduce
LFLAGS= -L$(QTDIR)/lib -L$(KDEDIR)/lib -L/usr/X11R6/lib
LIBS= -lqt -lX11 -lXext
CC=g++
helloworld: helloworld.o
$(CC) $(LFLAGS) -o helloworld helloworld.o $(LIBS)
helloworld.o: helloworld.cpp
clean:
rm -f helloworld
rm -f helloworld.o
.SUFFIXES: .cpp
.cpp.o:
$(CC) -c $(CFLAGS) $(INCL) -o $@ $<
```

makefile은 환경변수 QTDIR와 KDEDIR가 Qt와 KDE개발체계의 설치등록부이름으로 정의되는것으로 가정한다. 보통 이 2개의 환경변수는 쏘프트웨어를 설치할 때 설정된다. 표 2-1과 같이 makefile에서 5개의 이름이 정의된다.

표 2-1. Makefile에서 정의된 변수들

이름	내용
INCL	머리부파일들이 있는 위치의 경로이름. 이것은 머리부파일들을 찾는 위치를 콤파일러에 넘긴다. 항상 콤파일러는 표준머리부를 /usr/include에서 조사한다.
CFLAGS	콤파일러에 넘긴 옵션목록. -pipeoption은 2단계의 콤파일사이에 자료를 넘길 때 일시파일대신에 파이프들을 사용할것을 콤파일러에 지시한다. -O2옵션은 아주 높은 준위의 최적화를 지정한다. fno-strength-reduce옵션은 최적화가 순환변수들을 줄이거나 제거하는것을 막는다.
LFLAGS	련결프로그람에 넘기는 옵션목록. 매개 –Loption은 둘이상의 서고를 포함하

이름	내용
	고있는 등록부를 지정한다.
LIBS	이 프로그람에 요구되는 서고이름들의 목록. 이름있는 서고는 LFLAGS에 의해서 지정된 등록부들에서 탐색된다. 매개의 이름은 서고파일의 이름으로 확장된다. 실례로 -lqt는 libqt.so로 바뀌고 -lX11은 libX11.so로 된다.
CC	콤파일러의 이름.

makefile의 마지막 두 행은 .cpp파일을 .ofile로 바꾸는 지령을 구성하는 방법을 make에 알리는데 사용된다. 이 실례에는 원천파일이 하나만 있지만 여러개이면 변환규칙을 사용하여 콤파일지령을 한번 정의하여 전체 makefile에 적용하도록 한다.

알아두기: makefile을 쓰는 방법은 무한히 많다. 이 실례는 상대적으로 간단하므로 리해하기 쉽다. 응용프로그람을 개발할 때 makefiles에 다른것들도 추가해야 한다.

제2절. Hello KDE

그림 2-2에 보여주는것처럼 이 실례는 QApplication객체가 아니라 KApplication객체에 기초하고있는것을 제외하면 앞의 실례와 같다. KApplication클라스는 QApplication에 기초하므로 형식과 주제, KDE창문부품들의 사용능력, 표준KDE환경구성에 대한 호출, 쎄숀관리정보호출 그리고 사용자의 웨브열람기와 전자우편의뢰기호출과 같은 KDE기능들의 추가외에 근본적인 차이는 없다.

```
1 /* hellokde.cpp */
2 #include <kapplication.h>
3 #include <qlabel.h>
4 #include <qstring.h>
5
6 int main(int argc, char **argv)
7 {
8       KApplication app(argc, argv, "hellokde");
9       QLabel *label = new QLabel(NULL);
10      QString string("Hello, KDE");
11      label->setText(string);
12      label->setAlignment(
13              Qt::AlignVCenter | Qt::AlignHCenter);
14      label->setGeometry(0,0,180,75);
15      label->show();
16      app.setMainWidget(label);
17      return(app.exec());
18 }
```

그림 2-2. 본문을 현시하는 간단한 KDE프로그람

KApplication객체는 2행에서 포함한 머리부파일 kapplication.h에서 선언된다. kapplication.h파일은 qapplication.h파일을 포함하므로 Qt프로그람에 사용할수 있는 모든 기능은 KDE프로그람에도 사용할수 있다. 3행과 4행에 포함한 머리부파일들은 QLabel과 QString클라스의 선언을 보관한다.

8행에서 지령행인수와 응용프로그람이름을 넘기여 KApplication객체를 만든다. 응용프로그람이름은 그림기호들의 위치지정, 통보문수신 및 환경구성정보의 읽기와 같은 응용프로그람에 고유한 과제들에 사용될수 있다.

이 프로그람에서는 KDE객체를 사용하므로 객체를 보관하는 KDE서고를 포함하여야 한다. 전문화된 KDE서고가 있고 중요한 2개 서고는 libkdecore와 libkdeui이다.

```
INCL= -I$(QTDIR)/include -I$(KDEDIR)/include
CFLAGS= -O2 -fno-strength-reduce
LFLAGS= -L$(QTDIR)/lib -L$(KDEDIR)/lib -L/usr/X11R6/lib
LIBS= -lkdecore -lkdeui -lqt -lX11 -lXext -ldl
CC=g++
hellokde: hellokde.o
$(CC) $(LFLAGS) -o hellokde hellokde.o $(LIBS)
hellokde.o: hellokde.cpp
clean:
rm -f hellokde
rm -f hellokde.o
.SUFFIXES: .cpp
.cpp.o:
$(CC) -c $(CFLAGS) $(INCL) -o $@ $<
```

LIBS정의는 KDE의 핵심기능이 들어있는 서고 libkdecore.a와 KDE창문부품들이 모두 들어있는 libkdeui.a를 포함한다는것을 보여준다. KDE는 ODBC구동프로그람들을 동적적재하여 내부적으로 ODBC(Open Database Connectivity)를 실현하므로 서고 libdl.a를 포함하여야 한다. KDE를 설치할 때 이 서고를 기정등록부에 배치하므로 LFLAGS에 새로운 탐색경로를 추가할 필요는 없다.

제3절. 간단한 창문클라스

다음 실례는 자체의 창문부품을 만드는 기본형식을 보여준다. 이 프로그람은 그림 2-3에 보여주는것처럼 MyLabel창문부품을 만들어 기본창문에 현시한다. MyLabel창문부품은 매

우 간단하다. 그것은 QLabel의 모두를 계승하며 어떠한 기능도 추가하지 않는다. 클라스선언은 머리부파일 mylabel.h에 있다.

```
1 /* mylabel.h */
2 #ifndef MYLABEL_H
3 #define MYLABEL_H
4
5 #include <qlabel.h>
6 #include <qstring.h>
7
8 class MyLabel: public QLabel
9 {
10 public:
11     MyLabel(QWidget *parent);
12     ~MyLabel();
13 };
14
15 #endif
```
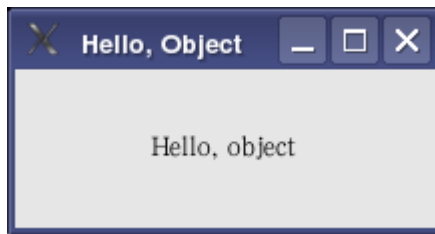
그림 2-3. 창문부품의 창조와 현시

2, 3 및 15행의 앞처리지령들은 반드시 필요한것은 아니지만 아주 편리하다. 응용프로그람이 더 복잡해질 때 다른 머리부파일들안에 머리부파일들을 포함하는것이 일반적이므로 같은 머리부파일이 하나의 원천파일에 여러번에 걸쳐 포함될수 있다. MYLABEL_H를 정의함으로써 이 머리부는 여러번 포함될수 있지만 오직 한번 콤파일된다.

8행에서 MyLabel클라스의 선언은 기초클라스로서 QLabel을 사용한다. 이것은 5행의 include지령이 QLabel의 선언을 유효하게 만들기 위한것이다. 머리부파일 qstring.h는 QString의 선언이 MyLabel구성자에 요구되므로 6행에 포함한다. MyLabel클라스는 자기의 원천파일에 실현된다. MyLabel의 구성자로부터 QLabel기초클라스의 구성자에로 부모창문부품의 주소를 넘기는것외에 다른 일을 하지 않는다.

```
1 /* mylabel.cpp */
2 #include "mylabel.h"
3
4 MyLabel::MyLabel(QWidget *parent) : QLabel(parent)
5 {
6 }
7 MyLabel::~MyLabel()
8 {
9 }
```

다음의 실례는 MyLabel창문부품을 만들어 현시한다. 리용하는 객체를 제외하고 이 프로그람의 main()함수는 이전 실례와 거의 같다.

```
1 /* helloobject.cpp */
2 #include "mylabel.h"
3 #include <qapplication.h>
4
5 int main(int argc, char **argv)
6 {
7       QApplication app(argc, argv);
8       MyLabel *mylabel = new MyLabel(NULL);
9       QString string("Hello, object");
10      mylabel->setText(string);
11      mylabel->setAlignment(
12              Qt::AlignVCenter | Qt::AlignHCenter);
13      mylabel->setGeometry(0,0,180,75);
14      mylabel->show();
15      app.setMainWidget(mylabel);
16      return(app.exec());
17 }
```

MyLabel객체는 이전 실례에서 QLabel객체와 같은 방법으로 정의되고 조작되고 현시된다. setAlignment()함수는 QLabel로부터 직접 계승되고 setGeometry()와 show()는 QWidget로부터 계승된다.

알아두기: 객체지향프로그람작성은 많은 우점을 가지고 있으나 가장 잘 알려진것은 도형방식사용자대면부(GUI)의 우점들이다. Qt는 이러한 우점을 리용한다. 현시가능한 모든 객체는 기초클라스 QWidget로부터 기본기능을 계승한다. 그것은 현시가능한 모든 창문 즉 표식자, 단추, 제일웃준위창문, 기타 모두가 크기, 색, 유표모양, 마우스탐지와 스크롤 등을 조종하는 기본함수들의 꼭같은 모임을 가진다는것을 의미한다. 이것은 자체의 창문부품들을 쉽게 만들뿐아니라 자동적으로 화면우의 모든것에 대하여 통일적인 기정동작과 외관을 적용할수 있게 한다.

makefile은 이전의 실례와 거의 같지만 두개의 개별적인 .cpp파일들을 콤파일하고 모두 련결한다.

```
INCL= -I$(QTDIR)/include -I$(KDEDIR)/include
CFLAGS= -pipe -O2 -fno-strength-reduce
LFLAGS= -L$(QTDIR)/lib -L$(KDEDIR)/lib -L/usr/X11R6/lib
LIBS= -lqt -lX11 -lXext
CC=g++
helloobject: helloobject.o mylabel.o
$(CC) $(LFLAGS) -o helloobject helloobject.o \
mylabel.o $(LIBS)
helloobject.o: helloobject.cpp mylabel.h
mylabel.o: mylabel.cpp mylabel.h
```

```
clean:
rm -f helloobject
rm -f mylabel.o
rm -f helloobject.o
.SUFFIXES: .cpp
.cpp.o:
$(CC) -c $(CFLAGS) $(INCL) -o $@ $<
```

제4절. 복합창문부품

QApplication객체는 assignMainWidget()메쏘드의 호출에 의해 할당되는 창문부품을 응용프로그람의 기본창문으로서 현시한다. 둘이상의 항목을 포함하는 기본창문을 현시하기 위하여 자체의 창문부품을 만들고 그것을 사용하여 기본창문으로 현시한다. 다음의 실례는 두 단추와 표식자를 하나의 창문부품에 결합한다.

```
1 /* threewidget.h */
2 #ifndef THREEWIDGET_H
3 #define THREEWIDGET_H
4
5 #include <qpushbutton.h>
6 #include <qlabel.h>
7
8 class ThreeWidget: public QWidget
9 {
10 public:
11     ThreeWidget(QWidget *parent=0,const char *name=0);
12 private:
13     QPushButton *topButton;
14     QPushButton *bottomButton;
15     QLabel *label;
16 };
17
18 #endif
```

5행과 6행은 합성창문부품에 포함되여야 할 창문부품들을 선언하는 머리부파일들을 포함한다. 거기에 2개의 단추와 하나의 표식자가 있다. 그리고 그것들의 주소를 보관하는 위치들은 threewidget.h의 13～15행에서 비공개자료로서 정의된다.

```
1 /* threewidget.cpp */
2 #include "threewidget.h"
3
4 ThreeWidget::ThreeWidget(QWidget *parent,const char *name):
5 QWidget(parent,name )
6 {
7     setMinimumSize(120,180);
8     setMaximumSize(120,180);
```

```
9
10    topButton = new QPushButton("Top Button",this);
11    topButton->setGeometry(15,15,90,40);
12    label = new QLabel("Middle Label",this);
13    label->setGeometry(15,70,90,40);
14    label->setAlignment(AlignVCenter | AlignHCenter);
15    bottomButton = new QPushButton("Bottom Button",this);
16    bottomButton->setGeometry(15,125,90,40);
17 }
```

ThreeWidget클라스는 현시가능한 창문부품이여야 하므로 기초클라스로서 QWidget를 사용한다.

7행과 8행은 창문부품의 최소 및 최대크기를 설정한다. 부모창문은 그것이 포함하는 창문부품이 크기를 설정할것을 요구한다. 이 실례에서 최소 및 최대의 설정값은 창문크기가 고정된다는것을 의미한다. 그림 2-4에 보여주는 현시창문은 마우스로 너비와 높이를 변경할수 없다.
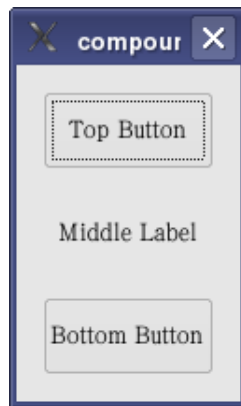
그림 2-4. 단추와 표식자들의 위치와 크기지정

제일 우의 단추는 10행에서 만들어진다. 구성자의 둘째 인수는 단추의 부모창문부품이다. 이 실례에서 부모는 구성하는 새로운 창문부품이다. 같은 부자관계는 12행과 15행에서 표식자와 다른 단추에도 수립된다.

현재 만든 창문부품은 120화소너비와 180화소높이의 현시가능구역을 가진다. 창문부품들은 setGeometry()호출에 의해서 창문에 배치된다. 11행에서 setGeometry()호출은 웃끝으로부터 15화소, 왼쪽에서 15화소에 배치하며 80화소너비와 40화소높이로 단추를 설정한다. 마찬가지로 13과 16행에서 setGeometry()호출은 다른 창문부품들을 배치한다. setGeometry()에 대한 첫 2개 인수는 왼쪽웃구석이고 2번째의 2개 인수는 너비와 높이이다.

이 프로그람의 main()함수는 새로운 합성창문부품을 마치도 다른 창문부품인것처럼 취급한다.

```
1 /* compound.cpp */
2 #include <qapplication.h>
3 #include "threewidget.h"
```

```
4
5 int main(int argc, char **argv)
6 {
7       QApplication app(argc, argv);
8       ThreeWidget threeWidget;
9       threeWidget.setGeometry(10,10,100,100);
10      app.setMainWidget(&threeWidget);
11      threeWidget.show();
12      return(app.exec());
13 }
```

threeWidget객체는 8행에서 만들어진다. 창문부품은 유효하지 않은 크기에 맞추도록 강요할수 없으므로 9행의 setGeometry()호출은 창문부품에 최소 및 최대크기설정으로 인한 영향이 없다. 11행에서 show()호출은 창문부품과 그것이 포함하는 모든 창문부품들을 현시하도록 지시한다.

제5절. 단추

단추는 창문부품이므로 다른 창문부품처럼 현시될수 있다. 그러나 프로그람은 언제 사용자가 단추를 찰칵하는지 알고있어야 한다. 다음의 실례는 그림 2-5에 보여주는 창문을 현시하고 단추사건에 응답하여 완료한다.

```
1 /* exitbutton.cpp */
2 #include <qapplication.h>
3 #include <qpushbutton.h>
4 #include <qstring.h>
5
6 int main(int argc, char **argv)
7 {
8       QApplication app(argc, argv);
9       QString string("Exit");
10      QPushButton *button = new QPushButton(string,NULL);
11      QObject::connect(button,
12              SIGNAL(clicked()),&app,SLOT(quit()));
13      button->setGeometry(0,0,80,50);
14      button->show();
15      app.setMainWidget(button);
16      return(app.exec());
17 }
```

그림 2-5. 프로그람을 완료하는 단추

단추는 마우스사건에 응답할 준비가 되여있지만 단추가 프로그람의 어떤 부분에 통보문을 보내도록 지시하지 않는 한 응답사건이 발생하지 않는다. 이런 형의 통보문을 신호라고 부르며 신호를 받아들이는 메쏘드를 처리부라고 한다. 11행과 12행에서 QObject::connect()에 대한 호출은 신호의 사본이 단추안의 clicked()로부터 응용프로그람의 quit()메쏘드로 보내지게 한다.

알아두기: 다른 사건구동체계에서 작업한적이 있다면 아마 역호출함수개념에 익숙되였을것이다. 처리부는 역호출과 류사하지만 차이가 있다. 가장 중요한 차이는 처리부들이 형안전한것이다. 인수형들이 서로 조화되지 않으면 프로그람은 콤파일되지 않는다.

QObject::connect()호출에서 단추의 clicked()메쏘드를 신호의 원천으로 지정한다. 신호는 그것을 받을 처리부함수모임이 있든 없든 발송된다. 또한 여러개의 처리부들이 신호를 받도록 설정되였으면 그것들은 각각 신호의 사본을 받는다.

QObject::connect()호출에서 둘째 쌍의 인수들은 받는 처리부가 QApplication의 quit()이라는것을 지정한다.

제6절. 신호용처리부의 정의

창문부품이 신호를 받도록 하기 위하여서는 처리부를 선언한 다음 신호에 련결해야 한다. 그림 2-6에서 보여주는 다음 실례는 단추와 계수기를 현시하고 단추를 누를 때마다 계수기를 증가시킨다. 이것을 발생시키는데 몇가지 필요한것이 있지만 Qt체계가 그 세부분을 거의 조종한다. 특히 세부적인 작업을 자동적으로 처리하기 위한 특수마크로들과 메타객체콤파일러(Meta Object Compiler, MOC)가 있다. 이 실례의 main()함수는 간단히 창문부품을 만들고 현시한다.

```
1 /* count.cpp */
2 #include <qapplication.h>
3 #include "clickcount.h"
4
5 int main(int argc, char **argv)
6 {
7     QApplication app(argc, argv);
8     ClickCount clickcount;
9     app.setMainWidget(&clickcount);
10    clickcount.show();
11    return(app.exec());
12 }
```
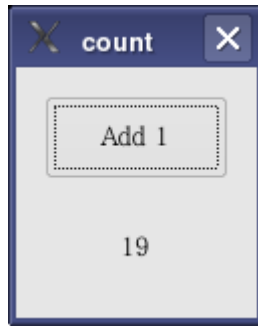
그림 2-6. 단추의 신호를 받는 계수기의 처리부

ClickCount창문부품은 단추와 표식자를 포함한다. 표식자는 현재계수기값을 현시하는데 쓰인다.

```
1 /* clickcount.h  */
2  #ifndef  CLICKCOUNT_H
3  #define  CLICKCOUNT_H
4
5  #include  <qpushbutton.h>
6  #include  <qlabel.h>
7
8  class  ClickCount:  public  QWidget
9  {
10      Q_OBJECT
11  public:
12      ClickCount(QWidget  *parent=0,const  char  *name=0);
13  public  slots:
14      void  incrementCounter();
15  private:
16      int counter;
17      QLabel *label;
18      QPushButton *button;
19 };
20
21 #endif
```

10행의 마크로 Q_OBJECT는 처리부를 가지는 모든 클라스에 존재해야 한다. (또한 5장에서 볼수 있는것처럼 신호를 발송하는 클라스에도 있어야 한다.) Q_OBJECT마크로는 신호와 처리부들이 작업하는데 필요한 표준메쏘드들을 정의한다.

14행에서 incrementCounter()메쏘드는 13행에서 공개처리부로 된다. incrementCounter()는 처리부로 선언되지만 클라스의 다른 메쏘드와 같이 신호로도 호출될수 있고 직접 호출될수도 있다.

ClickCount클라스의 구성자는 단추와 표식자를 포함하는 배치를 만들고 단추사건으로부터 incrementCounter()라는 이름의 처리부에 신호를 보내는 련결을 만든다.

```
1  /*  clickcount.cpp  */
```

```
2 #include <stdio.h>
3 #include "clickcount.h"
4
5 ClickCount::ClickCount(QWidget *parent,const char *name) :
6     QWidget(parent,name )
7 {
8     setMinimumSize(120,125);
9     setMaximumSize(120,125);
10
11    counter = 0;
12    button = new QPushButton("Add 1",this);
13    button->setGeometry(15,15,90,40);
14    label = new QLabel("0",this);
15    label->setGeometry(15,70,90,40);
16    label->setAlignment(AlignVCenter | AlignHCenter);
17
18    QObject::connect(
19            button,SIGNAL(clicked()),
20            this,SLOT(incrementCounter()));
21 }
22 void ClickCount::incrementCounter()
23 {
24    char str[30];
25    sprintf(str, "%d",++counter);
26    label->setText(str);
27 }
```

8행과 9행에서 setMinimumSize()과 setMaximumSize()호출은 120×125화소로 창문크기를 고정시킨다. 계수기값은 11행에서 초기화되고 단추와 표식자는 12～16행에서 정의된다.

18행에서 QObject::connect()호출은 신호에 처리부를 련결한다. 19행의 처음 2개 인수는 신호의 원천이 clicked()메쏘드라는것을 지정한다. clicked()신호는 pressed(), released() 및 toggled()라는 신호들과 함께 단추클라스의 성원이다.

참고: 5장에서 신호메쏘드를 만드는 실례들을 볼수 있다.

20행에서 QObject::connect()인수들의 2번째 쌍은 신호를 받는 객체와 메쏘드를 지정한다. 객체는 this(ClickCount의 현재 실례)이고 메쏘드는 incrementCounter()이다. 단추사건에 의해 clicked()신호가 발송될 때마다 incrementCounter()메쏘드에 의해 받아들이고 현시되는 값에 1을 더하고 단추의 본문을 갱신한다.

알아두기: 신호와 처리부사이에는 사실상 련결이 없다. 신호는 받아들이는 처리부가 있든 없든 발송되며 신호를 받아들이는 처리부는 여러개 될수도 있다. 또한 처리부는 임의의 개수의 신호를 받아들이도록 설정될수 있다.

이 실례의 makefile은 메타객체콤파일러가 클라스를 선언하는 머리부파일의 원천코드로서 입력을 처리하고 콤파일하여 프로그람과 련결할 새 원천파일을 생성한다.

```
1 INCL= -I$(QTDIR)/include -I$(KDEDIR)/include
2 CFLAGS= -O2 -fno-strength-reduce
3 LFLAGS= -L$(QTDIR)/lib -L$(KDEDIR)/include -L/usr/X11R6/lib
4 LIBS= -lqt -lX11 -lXext
5 CC=g++
6
7 count: count.o clickcount.o moc_clickcount.o
8 $(CC) $(LFLAGS) -o count count.o clickcount.o \
9 moc_clickcount.o $(LIBS)
10
11 count.o: count.cpp clickcount.h
12 clickcount.o: clickcount.cpp clickcount.h
13 moc_clickcount.cpp: clickcount.h
14 $(QTDIR)/bin/moc clickcount.h -o moc_clickcount.cpp
15
16 clean:
17 rm -f count
18 rm -f count.o
19 rm -f clickcount.o
20 rm -f moc_*
21
22 .SUFFIXES: .cpp
23
24 .cpp.o:
25 $(CC) -c $(CFLAGS) $(INCL) -o $@ $<
```

7행은 프로그람이 count.o와 clickount.o에 의존할뿐아니라 moc_clickcount.o에도 의존한다는것을 보여준다. 파일 moc_clickcount.cpp는 clickcount.h로부터 MOC콤파일러에 의해 만들어진다. Q_OBJECT마크로는 ClickCount클라스선언에 일부 메쏘드원형선언을 추가하고 MOC콤파일러는 새로운 메쏘드의 본체들을 생성한다. 결과 신호와 처리부에 요구되는 반복(그리고 오유판정)코드작성이 거의 완전히 자동화된다.

참고: 5장에서 전체 과정을 시험한다.

요 약

아주 적은 코드행으로 Qt 또는 KDE응용프로그람을 작성할수 있다. 창문을 만드는 코드작성의 세부와 사용자입력을 받아들이는 저준위기구는 모두 API내에서 조종된다.

· Qt프로그람은 프로그람의 기본함수에서 QApplication객체를 만들고 창문을 조종하도록 작성한다. KDE프로그람도 마찬가지로 KApplication객체를 사용하여 작성한다.

· 응용프로그람의 기본창문은 하나의 창문부품이다. 이 창문부품은 보통 정보를 현시하고 사용자대면부를 제공하는 다른 창문부품들의 모임을 포함한다.

· 객체는 하나이상의 신호를 발생할수 있도록 작성될수 있다. 또한 객체는 발생된 신호

를 받아들이도록 설계된 하나이상의 처리부를 포함할수 있다. 처리부와 신호실현의 세부는 마크로들과 MOC콤파일러의 사용을 통하여 자동화된다.

이 장은 응용프로그람의 기본창문을 만들고 현시하는 방법을 설명하였다. 다음 장은 튀여나오기창문과 대화칸을 현시하는 방법을 취급한다. 미리 정의된 KDE와 Qt대화칸들이 있지만 자체로 창조할수도 있다.

제3장. 창문의 창문부품배치

학습내용

현시기에서 x, y값을 리용한 창문부품들의 위치지정

살창자리표계와 창문부품들의 결합

현시기에 적합하게 창문부품들의 확장과 축소

수직칸에 창문부품들의 쌓기

수평칸에 창문부품들의 삽입하는 방법

이 장은 창문내부에서 창문부품집합의 크기와 위치를 조종하는 방법을 설명한다. 응용프로그람을 시작할 때 단추와 본문입력창문부품들을 적당히 배렬하여 표시하여야 단추사건에 응답하거나 사용자가 입력한 본문을 읽어들일수 있다. 이 장은 창문부품을 필요한 위치에 배치하는 방법뿐아니라 창문부품크기와 창문크기가 달라질 때 발생하는 작용을 지정하는 방법을 설명한다.

창문부품들은 자리표값들에 의하여 혹은 배치를 리용하여 위치와 크기를 지정할수 있다. 자리표값들은 사용자가 조절할수 없는 고정된 화소위치들이다. 한편 배치객체는 창문부품들호상 및 포함하는 창문의 전체 크기에 따라 최대 및 최소한계내에서 창문부품들의 위치와 크기를 조종한다.

제1절. 창문부품의 기하학적배치

4개의 값 수평변이, 수직변이, 너비와 높이를 지정함으로써 매개 창문부품의 정확한 배치와 크기를 지정할수 있다. 자리표계는 부모창문 즉 창문부품들을 포함하는 창문의 자리표계이다. 다음 실례는 응용프로그람창문에 3개의 누름단추를 배치한다.

Main

```
1 /* main.cpp */
2 #include <kapplication.h>
3 #include "setxy.h"
4
5 int main(int argc, char **argv)
6 {
7     KApplication app(argc, argv, "setxy");
8     SetXY setxy;
9     setxy.show();
10    app.setMainWidget(&setxy);
11    return(app.exec());
12 }
```

기본함수는 아주 간단한다. 머리부파일은 3행에서 포함된다. 8행에서 단추들을 가지는 창문부품이 만들어지고 10행에서 setMainWidget()호출에 의해 기본응용프로그람창문으로 현시되는 창문부품으로 설정된다.

SetXY머리부파일

```
1 /* setxy.h */
2  #ifndef  SETXY_H
3  #define  SETXY_H
4
5  #include  <qpushbutton.h>
6
7  class  SetXY:  public  QWidget
8  {
9  public:
10       SetXY(QWidget  *parent=0,const  char  *name=0);
11       ~SetXY();
12  private:
13       QPushButton  *button1;
14       QPushButton  *button2;
15       QPushButton *button3;
16 };
17
18 #endif
```

클라스 SetXY는 QWidget를 기초클라스로 하여 선언된다. 자료는 13～15행에서 정의된 3개의 QPushButton지적자이다. 보통 단추사건들에 응답하는 처리부로 지정되는 메쏘드가 있을수 있지만 이것은 간단한 배치를 보여주는 실례이므로 단추사건들에 응답하지 않는다.

> 알아두기: setxy.h의 5행에서 포함되는 머리부파일 qpushbutton.h는 자동적으로 setxy.cpp와 main.cpp에도 포함된다. 또 하나의 창문부품을 사용하는데 그 창문부품도 역시 qpushbutton.h를 포함한다면 같은 머리부파일은 두번 포함된다. 2, 3, 18행의 사전콤파일러지령들은 머리부파일이 여러번 콤파일되는것을 막는데 쓰인다.

SetXY

```
1  /*  setxy.cpp  */
2  #include  "setxy.h"
3
4  SetXY::SetXY(QWidget  *parent,const  char  *name)
5  :  QWidget(parent,name)
6  {
7       setMinimumSize(90,40);
```

```
8       setMaximumSize(190,220);
9       resize(190,220);
10
11      button1  =  new  QPushButton("Upper  Left",this);
12      button1->setGeometry(0,0,90,40);
13
14      button2  =  new  QPushButton("Middle  Right",this);
15      button2->setGeometry(90,70,100,50);
16
17      button3  =  new  QPushButton("Bottom",this);
18      button3->setGeometry(45,140,50,80);
19 }
20 SetXY::~SetXY() { }
```

SetXY가 수행하는 모든 작업은 구성자에서 진행된다. 그림 3-1에서 보여주는것처럼 3개의 단추가 정의되고 현시된다.
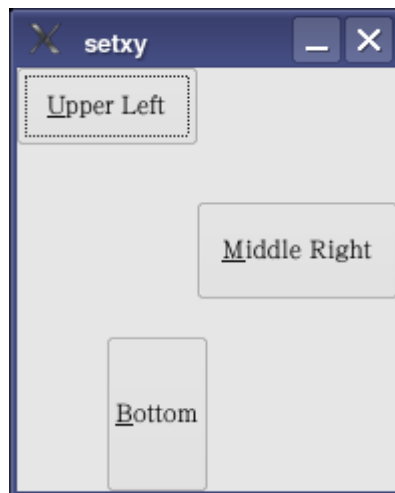
그림 3-1. 자리표에 따라 배치된 3개의 단추들

11행에서 왼쪽웃구석에 있는 단추가 만들어지고 12행에서 그 크기와 위치가 설정된다. setGeometry()에 넘긴 4개 옹근수의 순서는 다음과 같다.

x , y , width , height

거리는 화소로 계산된다. x값은 응용프로그람창문의 왼끝으로부터 포함된 창문부품의 왼끝까지의 화소수이다. y값은 창문의 웃끝으로부터 창문부품의 웃끝까지의 화소수이다. 즉 왼쪽웃구석은 자리표계의 원점이다. 매개 단추를 만든 다음 setGeometry()호출에 의해 크기와 위치가 설정된다.

알아두기: 2개 창문부품이 같은 공간을 차지하도록 배치하면 새로운 창문부품이 낡은 창문부품을 가리운다.

7, 8, 9행의 호출은 창문부품을 현시하는 규칙들을 정의한다. 7행의 setMinimumSize()호출은 창문부품이 적어도 90×40화소 크기로 현시되여야 한다는것을 지정한다. 창문부품은 사용자가 설

정한 크기로 창문에 현시되므로 최소설정값은 중요하다. 창문부품은 그림 3-2에서 보여주는 크기이상으로 줄어들수 없다. 8행의 setMaximumSize()호출은 창문부품의 높이와 너비에 대한 웃한계를 설정한다. 끝으로 9행의 resize()호출은 창문부품의 초기크기를 최대허용크기로 설정한다.
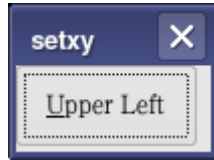
그림 3-2. 3개 단추중 2개를 볼수 없게 하는 최소크기설정

20행의 ~SetXY()해체자에서는 응용프로그람은 완료되고 창문부품은 파괴되므로 아무것도 수행할것이 없다. 또한 이 창문부품이 단추들의 부모이므로 3개 단추를 해체한다.

제2절. 창문부품의 크기

창문에서 창문부품들을 배치할 때 2가지 중요한 인자를 고려해야 한다. 매개 창문부품은 위치와 크기를 가지고있다. 이전 실례에서 위치와 크기는 응용프로그람에 의해 완전히 조종된다. 응용프로그람이 자세한 조종준위를 요구하면 좋지만 대부분의 창문부품들은 크기에 대한 자기의 요구를 가지고있다.

QWidget의 수많은 메쏘드들은 창문부품의 크기에 대한 조종을 사용자에게 넘겨준다. 이러한 메쏘드들의 일부는 높이와 너비값을 사용하고 일부는 QSize객체를 사용하지만 그것들은 모두 같은 일을 한다. QSize클라스는 간단히 높이와 너비를 포함하지만 크기조작이 복잡해질 때 조작을 훨씬 더 쉽게 하는 메쏘드와 연산자들도 포함한다. 실례로 비례에 따라 크기를 변경하는 연산자와 두개의 QSize객체를 하나로 결합하는 연산자를 리용할수 있다.

```
void setMaximumSize(const QSize &qsize);
void setMaximumSize(int width,int height);
void setMaximumWidth(int width);
void setMaximumHeight(int height);
void setMinimumSize(const QSize &qsize);
void setMinimumSize(int width,int height);
void setMinimumWidth(int width);
void setMinimumHeight(int height);
```

창문부품이 고정크기를 가져야 한다면 2개의 함수를 만들어 최소값과 최대값을 같은 값들로 설정하거나 다음 함수들중 하나를 호출하여 설정할수 있다.

```
void setFixedSize(const QSize &qsize);
void setFixedSize(int width,int height);
void setFixedWidth(int width);
void setFixedHeight(int height);
```

다음의 함수들은 최소크기와 최대크기를 얻는다.

```
QSize maximumSize();
QSize minimumSize();
```

제3절. 고정살창창문부품배치

QGridLayout객체의 사용은 보이지 않는 수평선과 수직선들로 이루어진 살창을 정의하고 세포들에 창문부품들을 삽입할수 있게 한다. 다음 실례는 5×5세포 크기로 살창을 만들고 4개 세포에 단추를 삽입한다.

Main

```
1 /* main.cpp */
2 #include <kapplication.h>
3 #include "fivebyfive.h"
4
5 int main(int argc, char **argv)
6 {
7      KApplication app(argc, argv, "fivebyfive");
8      FiveByFive *fivebyfive = new FiveByFive();
9      fivebyfive->show();
10     app.setMainWidget(fivebyfive);
11     return(app.exec());
12 }
```

기본함수는 FiveByFive창문부품의 실례를 만들고 그것을 KApplication창문에 현시하는 창문부품으로 사용한다. 결과는 그림 3-3에 현시된 창문이다.
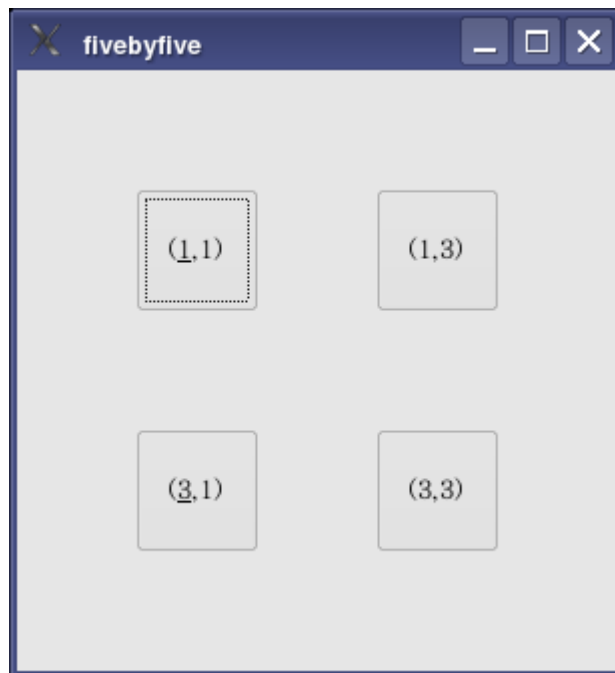
그림 3-3. 살창배치에 의한 4개 누름단추의 배치

FiveByFive머리부파일

```
1 /* fivebyfive.h */
2 #ifndef FIVEBYFIVE_H
3 #define FIVEBYFIVE_H
4
5 #include <qwidget.h>
6 #include <qpushbutton.h>
7
8 class FiveByFive: public QWidget
9 {
10 public:
11     FiveByFive(QWidget *parent=0,const char *name=0);
12     ~FiveByFive();
13 private:
14     QPushButton *b11;
15     QPushButton *b31;
16     QPushButton *b13;
17     QPushButton *b33;
18 };
19
20 #endif
```

머리부파일은 FiveByFive클라스를 QWidget의 파생클라스로 선언하고 살창에 4개 단추를 포함한다.

FiveByFive

```
1 /* fivebyfive.cpp */
2 #include <qlayout.h>
3 #include "fivebyfive.h"
4
5 FiveByFive::FiveByFive(QWidget *parent,const char *name)
6 : QWidget(parent,name)
7 {
8     QGridLayout *layout = new QGridLayout(this,5,5);
9
10    b11 = new QPushButton("(1,1)",this);
11    b11->setMaximumSize(100,100);
12    layout->addWidget(b11,1,1);
13    b13 = new QPushButton("(1,3)",this);
14    b13->setMaximumSize(100,100);
15    layout->addWidget(b13,1,3);
16    b31 = new QPushButton("(3,1)",this);
17    b31->setMaximumSize(100,100);
18    layout->addWidget(b31,3,1);
19    b33 = new QPushButton("(3,3)",this);
20    b33->setMaximumSize(100,100);
21    layout->addWidget(b33,3,3);
```

```
22
23      for(int  i=0;  i<5;  i++)  {
24              layout->addRowSpacing(i,60);
25              layout->addColSpacing(i,60);
26      }
27      resize(10,10);
28
29      layout->activate();
30 }
31
32 FiveByFive::~FiveByFive() { }
```

파일 qlayout.h는 2행에서 포함된다. 또한 이 머리부파일은 이 장의 뒤에 설명하는 QVBoxLayout와 QHBoxLayout를 선언하는데 사용된다.

8행에서 만들어진 QGridLayout객체는 5×5세포의 크기를 가진다. 왼쪽웃구석의 세포는 (0,0)이고 그 오른쪽이 (0,1), 다음것이 (0,2),…. 왼끝에서 하나 옆에, 우로부터 하나 아래에 놓이도록 창문부품을 살창에 배치하기 위하여 살창세포 (1,1)에 설정한다.

10행과 11행은 단추를 만들고 그 최소높이와 최소너비를 모두 100화소로 설정한다. 이것은 단추가 자기에게 할당된 세포가 100✕100화소보다 작지만 세포가 꽉 차도록 단추를 확장하게 한다. 보통 단추는 그 내부최대설정값이상 늘일수 없다.

10, 11, 12행은 표식자 "(1, 1)"을 가지는 단추를 만들어 위치 (1,1)의 살창세포에 배치한다. 11행의 setMaximumSize()호출은 단추가 들어있는 세포가 꽉차도록 단추를 확장한다. sizeHint()메쏘드는 모든 창문부품에 사용할수 있는것은 아니다. 이 경우에 QPushButton창문부품이 유효크기암시를 돌려주는것으로 정확히 가정되므로 그것이 사용되기전에 유효성검사가 실시되지 않는다. 일반화된 코드형식은 다음과 같다.

```
QSize  *qsize  =  b11.sizeHint();
if(qsize.isValid())
    b11.setMinimumSize(qsize);
else
    b11.setMinimumSize(30,30);
```

이리하여 검사하는 창문부품이 유효크기설정을 돌려주지 않으면 적당한 값이 사용된다. 기정설정은 매우 좋지만 결코 사용자가 요구하는 정확한 값은 아니다.

> 알아두기: 자체의 창문부품을 작성할 때 sizeHint()를 실현하는것이 효과적일수 있다. sizeHint()는 QWidget에서 가상함수로 선언되며 그것을 재정의하지 않는한 무효한 QSize를 돌려준다.

23～26행의 순환은 살창의 매개 행과 렬에 대하여 addRowSpacing()을 호출하여 매개 렬의 최소너비와 매개 행의 최소높이를 각각 60화소로 설정한다. 최대값을 설정하지 않으면 매개 행과 렬은 임의의 크기까지 늘어날수 있으며 창문크기를 변경하면 단추들의 크기도 변경된다.

31행의 resize()호출은 10×10화소로 창문부품을 줄일것을 요구하지만 창문부품은 행과

렬의 최소크기때문에 그 요구를 받아들일수 없다. 창문부품이 자체의 크기를 변경할데 대한 지령을 받았는데 새로운 높이나 너비가 최대 또는 최소 한계를 벗어날 때에 요구된 값은 무시되고 가장 가까운 유효값 즉 최대값 또는 최소값이 리용된다. 이 실례에서 창문부품은 간단히 그 최소크기로 줄어든다.

제4절. 가변살창창문부품배치

다음 실례는 QGridLayout객체를 사용하여 4개 단추를 배치하고 부모창문의 크기가 변경될 때 단추들의 크기를 조종하는 행과 렬의 가변값들을 설정한다.

Main

```
1 /* main.cpp */
2 #include <kapplication.h>
3 #include "fourbyfour.h"
4
5 int main(int argc, char **argv)
6 {
7     KApplication app(argc, argv, "fourbyfour");
8     FourByFour *fourbyfour = new FourByFour();
9     fourbyfour->show();
10    app.setMainWidget(fourbyfour);
11    return(app.exec());
12 }
```

기본함수는 FourByFour창문부품의 실례를 만들고 그것을 KApplication창문에 현시하는 창문부품으로 사용한다.

FourByFour머리부파일

```
1 /* fourbyfour.h */
2 #ifndef FOURBYFOUR_H
3 #define FOURBYFOUR_H
4
5 #include <qwidget.h>
6 #include <qpushbutton.h>
7
8 class FourByFour: public QWidget
9 {
10 public:
11    FourByFour(QWidget *parent=0,const char *name=0);
12    ~FourByFour();
13 private:
14    QPushButton *b11;
15    QPushButton *b21;
16    QPushButton *b12;
17    QPushButton *b22;
```

```
18 };
19
20
21 #endif
```

머리부파일은 크기를 조종하는 세포들에 배치할 4개의 단추들을 비공개자료로 포함하는 QWidget로서 FourByFour클라스를 선언한다.

FourByFour

```
1 /* fourbyfour.cpp */
2 #include <qlayout.h>
3 #include "fourbyfour.h"
4
5 FourByFour::FourByFour(QWidget *parent,const char *name)
6 : QWidget(parent,name)
7 {
8     QGridLayout *layout = new QGridLayout(this,4,4);
9     QSize buttonMax(400,400);
10
11    b11 = new QPushButton(this);
12    b11->setText("(1,1)");
13    b11->setMinimumSize(b11->sizeHint());
14    b11->setMaximumSize(buttonMax);
15    layout->addWidget(b11,1,1);
16    b12 = new QPushButton(this);
17    b12->setText("(1,2)");
18    b12->setMinimumSize(b12->sizeHint());
19    b12->setMaximumSize(buttonMax);
20    layout->addWidget(b12,1,2);
21    b21 = new QPushButton(this);
22    b21->setText("(2,1)");
23    b21->setMinimumSize(b21->sizeHint());
24    b21->setMaximumSize(buttonMax);
25    layout->addWidget(b21,2,1);
26    b22 = new QPushButton(this);
27    b22->setText("(2,2)");
28    b22->setMinimumSize(b22->sizeHint());
29    b22->setMaximumSize(buttonMax);
30    layout->addWidget(b22,2,2);
31
32    layout->addRowSpacing(0,20);
33    layout->addRowSpacing(3,20);
34    layout->addColSpacing(0,20);
35    layout->addColSpacing(3,20);
36    resize(10,10);
37
38    layout->setRowStretch(2,100);
```

```
39      layout->setColStretch(2,100);
40
41      layout->activate();
42 }
43
44 FourByFour::~FourByFour() { }
```

8행에서 4×4세포크기로 QGridLayout객체가 만들어진다. 11∼30행은 4개의 QPushButton 객체를 만들어 살창의 4개의 중심세포에 추가한다. 매개 단추의 최소크기는 본문을 항상 볼 수 있는 값으로 설정된다. 매개 단추의 최대크기는 세포들을 채우는데 필요한만큼 QGridLayout가 단추들을 늘일수 있도록 일정하게 큰수로 설정된다.

32∼35행의 addRowSpacing()과 addColSpacing()호출은 좌우측 렬들의 너비와 우아래행들의 높이를 20화소로 각각 설정한다. 결과는 단추들을 담고있는 응용프로그람창문의 경계와 중심의 4개 세포사이의 20화소여백이다. 36행의 resize()호출은 최소허용값보다 작은 크기로 창문부품을 줄이려고 하므로 창문부품은 처음에 최소허용크기로 현시된다. 결과는 그림 3-4에 보여준다.
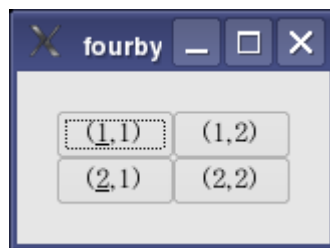
그림 3-4. 매개가 최소크기 4를 가지는 4개 창문부품

38행의 setRowStretch()호출은 3번째 행(단추들의 둘째 행)의 가변결수를 100으로 설정한다. 34행의 setColStretch()호출은 3번째 렬(단추들의 두번째 렬)의 가변결수를 100으로 설정한다. 창문의 수직크기 또는 수평크기가 바뀔 때마다 큰 가변결수를 가지는 행과 렬들은 작은 가변결수를 가지는것들보다 더 많이 변경된다.

기정가변값은 매개 행과 렬에 대하여 0이다. 행(또는 렬)들에 대하여 모두 0으로 설정되면 모든 행(또는 렬)의 크기를 변경할수도 있고 변경하지 않을수도 있다. 이 기정값은 대체로 사용자가 요구하는 값이 아니다. 살창의 크기가 달라질 때 령아닌 값을 가지는 행과 렬들만 그에 따라 크기가 달라진다. 행 또는 렬의 크기변경량은 그 행 또는 렬의 가변결수와 크기변경되는 모든 세포들의 가변결수들의 합의 비률에 의해 결정된다.

이 실례에서 창문을 확장하면 그림 3-5에 보여주는것과 같이 현시된다. 렬1과 행1의 가변결수들은 0으로 남아있으므로 왼쪽웃구석의 단추는 변경되지 않은 상태로 남아 있다. 다른 한편 렬2와 행2는 모두 령 아닌 값을 가지므로 오른쪽아래의 세포에 있는 단추는 량방향으로 크기가 변경된다.
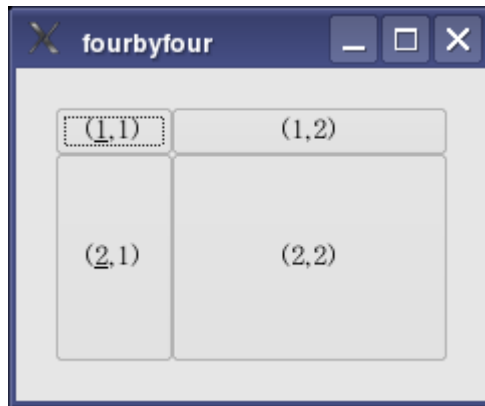
그림 3-5. 가변결수의 영향을 받은 각이한 크기변경

세포가 실제로 확장되는 크기는 세포의 가변결수와 변화되는 방향의 모든 가변결수들의 합의 비률에 의해 결정된다. 실례로 FourByFour의 가변결수변경은 모든 단추들의 크기를 변경하지만 일부는 다른것들보다 더 빨리 변경된다. 실례로 33행과 34행을 다음과 같이 교체한다.

```
layout->setRowStretch(1,50);
layout->setRowStretch(2,100);
layout->setColStretch(1,9);
layout->setColStretch(2,1);
```

이 환경설정값에 의하여 창문을 수직으로 늘이면 행1의 높이는 행2 보다 두배 증가한다. 창문을 수평으로 늘이면 렬1는 렬2의 9배로 증가한다. 창문확장결과는 그림 3-6에 보여준다.
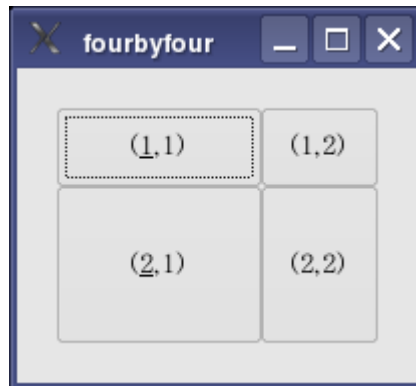
그림 3-6. 가변결수를 가지는 매개 행과 렬

제5절. 살창의 여러 세포를 차지하는 창문부품

QGridLayout는 장기판과 같은 직4사각형배렬에 창문부품들을 배치하는데 사용될수 있으며 창문부품이 2개이상의 바른4각형의 살창을 차지하게 할수 있다. 다음 실례는 표식자가 창문의 웃부분을 가로질러 적당히 놓이도록 하고 목록칸은 6개 세포의 위치를 차지하게 한다.

Main

```
1 /* main.cpp */
2 #include <kapplication.h>
3 #include "multicell.h"
4
5 int main(int argc, char **argv)
6 {
7     KApplication app(argc, argv, "multicell");
8     MultiCell *multicell = new MultiCell();
9     multicell->show();
10    app.setMainWidget(multicell);
11    return(app.exec());
12 }
```

MultiCell머리부파일

```
1 /* multicell.h */
2 #ifndef MULTICELL_H
3 #define MULTICELL_H
4
5 #include <qwidget.h>
6 #include <qpushbutton.h>
7 #include <qlabel.h>
8 #include <qlistbox.h>
9
10 class MultiCell: public QWidget
11 {
12 public:
13    MultiCell(QWidget *parent=0,const char *name=0);
14    ~MultiCell();
15 private:
16    QLabel *label;
17    QListBox *listbox;
18    QPushButton *addButton;
19    QPushButton *deleteButton;
20    QPushButton *cancelButton;
21 };
```

```
22
23 #endif
```

16～20행에서 서로 다른 모든 창문부품들의 지적자들을 MultiCell클라스의 자료성원으로 정의한다. 이것은 6～8행에 창문부품의 머리부파일이 있어야 한다는것을 의미한다.

알아두기: 클라스선언에서 모든 창문부품에 대하여 지적자를 포함하는것은 아주 례사로운 일이지만 늘 그러한것은 아니다. 일단 창문부품의 현시환경이 구성되였으면 어떤 구체적인 상황에서 지적자를 요구하지 않는한 프로그람이 그것을 기억할 필요는 없다. 실례로 자기 프로그람이 표식자본문을 변경하거나 목록칸에 성원들을 추가하려고 한다면 그 지적자들을 호출해야 한다.

MultiCell

```
1 /*  multicell.cpp  */
2 #include  <qlayout.h>
3 #include  <stdio.h>
4 #include  "multicell.h"
5
6 MultiCell::MultiCell(QWidget  *parent,const  char  *name)
7  :  QWidget(parent,name)
8  {
9      QGridLayout  *layout  =  new  QGridLayout(this,4,2,20);
10
11     label  =  new  QLabel("A  list  box  with  three  buttons",
12             this);
13     label->setMinimumSize(label->sizeHint());
14     label->setAlignment(AlignHCenter);
15     layout->addMultiCellWidget(label,0,0,0,1);
16
17     listbox  =  new  QListBox(this);
18     for(int  i=0;  i<20;  i++)  {
19             char  str[40];
20             sprintf(str,  "Selection  %d\n",i);
21             listbox->insertItem(str);
22     }
23     listbox->setMinimumWidth(120);
24     layout->addMultiCellWidget(listbox,1,3,0,0);
25
```

```
26     addButton  =  new  QPushButton(this);
27     addButton->setText("Add");
28     addButton->setMinimumSize(addButton->sizeHint());
29     layout->addWidget(addButton,1,1);
30
31     deleteButton  =  new  QPushButton(this);
32     deleteButton->setText("Delete");
33     deleteButton->setMinimumSize(deleteButton->sizeHint());
34     layout->addWidget(deleteButton,2,1);
35
36     cancelButton  =  new  QPushButton(this);
37     cancelButton->setText("Cancel");
38     cancelButton->setMinimumSize(cancelButton->sizeHint());
39     layout->addWidget(cancelButton,3,1);
40
41     resize(10,10);
42     layout->activate();
43 }
44
45 MultiCell::~MultiCell() { }
```

9행에서 QGridLayout는 4×2세포크기로 구성된다. 또한 경계값 20은 모든 세포들사이에 간격을 삽입하는데 사용된다. 현시된 창문을 그림 3-7에 보여준다.
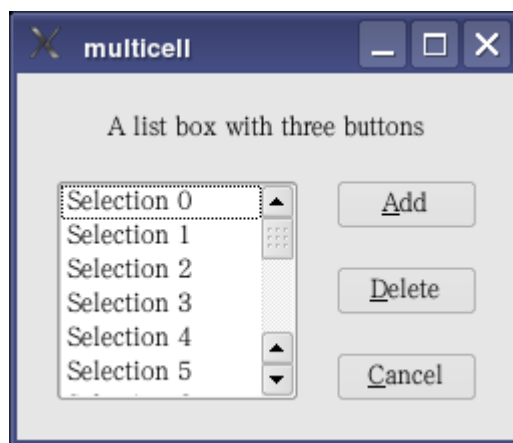
그림 3-7. 여러개의 살창세포들을 차지하는 창문부품들

11～14행은 본문을 가지는 표식자를 정의하고 최소크기를 설정하며 본문이 수평으로 중심에 놓이도록 배치한다. 15행에서 표식자는 addMultiCellWidget()호출에 의해 2개 세포에

배치된다. 처음 두 수는 행범위를, 두번째 두 수는 렬범위를 다음과 같이 지정한다

startRow, endRow, startCol, endCol.

17～24행은 20개 항목을 가진 목록칸을 만들고 최소너비를 120화소로 설정한다. 24행에서 addMultiCellWidget()호출은 목록칸이 6개의 세포 즉 2×3세포를 차지하도록 한다.

26～29행은 오른쪽에 3개 단추를 만들고 매개를 한세포에 하나씩 배치한다. 그림 3-7에 보여준 결과는 9행에서 모든 세포들에 지정한것처럼 20화소간격으로 구분된 단추들을 현시한다.

제6절. 수직칸배치

QVBoxLayout객체에 창문부품그룹을 삽입함으로써 수직렬에 창문부품그룹을 적당히 배치할수 있다. 처음에 삽입된 창문부품은 칸의 웃끝에 나타나고 2번째 창문부품은 그 아래에 배치되며 그 다음에 추가되는 창문부품은 목록의 제일 마지막 위치에 배치된다. 다음 실례는 수직칸에 5개 단추를 공간과 가변성을 조절하여 삽입한다.

Main

```
1 /* main.cpp */
2 #include <kapplication.h>
3 #include "verticalbox.h"
4
5 int main(int argc, char **argv)
6 {
7     KApplication app(argc, argv, "verticalbox");
8     VerticalBox *verticalbox = new VerticalBox();
9     verticalbox->show();
10    app.setMainWidget(verticalbox);
11    return(app.exec());
12 }
```

VerticalBox객체는 응용프로그람창문의 현시창문부품으로 사용된다.

VerticalBox머리부파일

```
1 /* verticalbox.h */
2 #ifndef VERTICALBOX_H
3 #define VERTICALBOX_H
4
5 #include <qwidget.h>
6 #include <qlayout.h>
7 #include <qpushbutton.h>
```

```
8
9 class VerticalBox: public QWidget
10 {
11 public:
12    VerticalBox(QWidget *parent=0,const char *name=0);
13    ~VerticalBox();
14 private:
15    QPushButton *buttonOne;
16    QPushButton *buttonTwo;
17    QPushButton *buttonThree;
18    QPushButton *buttonFour;
19    QPushButton *buttonFive;
20 };
21
22 #endif
```

VerticalBox

```
1 /* verticalbox.cpp */
2 #include "verticalbox.h"
3
4 VerticalBox::VerticalBox(QWidget *parent,const char *name)
5 : QWidget(parent,name)
6 {
7    QVBoxLayout *layout = new QVBoxLayout(this,5);
8    QSize buttonMaximum(400,400);
9
10   buttonOne = new QPushButton(this);
11   buttonOne->setText("BUTTON ONE");
12   buttonOne->setMinimumSize(buttonOne->sizeHint());
13   buttonOne->setMaximumSize(buttonMaximum);
14   layout->addWidget(buttonOne);
15
16   buttonTwo = new QPushButton(this);
17   buttonTwo->setText("BUTTON TWO");
18   buttonTwo->setMinimumSize(buttonTwo->sizeHint());
19   buttonTwo->setMaximumSize(buttonMaximum);
```

```
20  layout->addWidget(buttonTwo,30);
21
22  layout->addSpacing(20);
23
24  buttonThree = new QPushButton(this);
25  buttonThree->setText("BUTTON THREE");
26  buttonThree->setMinimumSize(buttonThree->sizeHint());
27  buttonThree->setMaximumSize(buttonMaximum);
28  layout->addWidget(buttonThree);
29
30  layout->addStretch(30);
31
32  buttonFour = new QPushButton(this);
33  buttonFour->setText("BUTTON FOUR");
34  buttonFour->setMinimumSize(buttonFour->sizeHint());
35  buttonFour->setMaximumSize(buttonMaximum);
36  layout->addWidget(buttonFour);
37
38  layout->addSpacing(5);
39  layout->addStretch(10);
40
41  buttonFive = new QPushButton(this);
42  buttonFive->setText("BUTTON FIVE");
43  buttonFive->setMinimumSize(buttonFive->sizeHint());
44  buttonFive->setMaximumSize(buttonMaximum);
45  layout->addWidget(buttonFive);
46
47  resize(10,10);
48  layout->activate();
49 }
50 VerticalBox::~VerticalBox() { }
```

이 클라스는 7행에서 만들어진 QVBoxLayout객체에 기초하는 창문부품이다. 구성자의 둘째 인수는 5화소경계선이 포함된 모든 항목주위에 삽입되도록 지정한다. 이 경계선이 0(기정)으로 설정되면 포함된 항목들은 서로 린접에 놓인다.

10～14행은 하나의 단추를 만들고 그 본문과 최소크기를 설정하고 그것을 배치에 추가

한다. 그 최대크기를 어떤 큰 값으로 설정하여 QVBoxLayout객체를 적당히 늘일수 있다. 그것은 배치에 추가된 첫번째 단추이므로 제일 우에 나타난다.

16~20행은 둘째 단추를 만들고 배치에 추가한다. 그림 3-8과 같이 첫째와 둘째 단추들사이의 거리는 10화소(단추1에서 5의 경계 + 단추1에서 5의 경계)이다. 그것들사이의 거리는 배치가 늘어날 때도 변화되지 않는다. 20행에서 가변결수 20은 단추가 칸에 추가될 때 지정되는데 이것은 배치크기가 변화할 때 단추자체가 가변되게 한다.

그림 3-8. 수직으로 가변되기전과 후의 수직칸

22행은 단추2아래에 20화소공간을 삽입한다. 그다음 24~28행에서 단추3을 만들고 배치에 추가된다. 이것은 단추2와 단추3사이가 항상 30화소(2개의 5화소경계 + 20화소공간)라는것을 의미한다.

30행은 단추3아래에 가변결수 30을 가지는 가변점을 추가하고 32~36행은 그 아래에 단추4를 삽입한다. 그림 3-8과 같이 배치가 최소크기로 되였을 때 가변결수에 의해 단추3과 4사이에 어떤 공간도 추가되지 않는다. 그러나 창문이 수직으로 가변될 때 두 단추사이에 공간이 나타난다. 38행은 단추4아래에 5화소공간과 가변결수10을 삽입한다. 그다음 41~45행은 그 아래에 단추5를 만들어 삽입한다. 이것은 두 단추사이의 최소공간이 15화소(2개의 5화소경계 + 5화소공간)이라는것을 의미한다.

가변량은 모든 가변결수들의 합과 매개 개별적 가변결수의 비이다. 이 실례는 가변결수 30(20행), 30(30행) 그리고 10(39행)을 각각 지정한다. 가변결수의 합은 70이므로 창문크기가 바뀔 때 단추2는 변화의 3/7을 흡수하고 단추2와 3사이의 공간은 3/7을 흡수하며 단추4와 5사이의 공간은 1/7을 흡수한다.

47행은 배치의 처음크기를 최소크기로 줄인다. 배치는 포함하고있는 모든 창문부품들과

공간의 최소크기를 합하여 최소크기를 결정한다. 48행은 부모창문이 현시될 때마다 배치가 실현되도록 한다.

제7절. 수평칸배치

수평칸은 수직칸에서와 같지만 포함된 창문부품들이 우에서 아래로 놓이지 않고 왼쪽에서 오른쪽으로 나란히 놓인다. 그림 3-9에 보여준 창문은 2가지 변경을 제외하고는 앞절에서 본 VerticalBox창문부품과 등가한 HorizontalBox창문부품에 의해 만들어진다. 수평칸의 구성자는 다음과 같다.

7 QHBoxLayout *layout = new QHBoxLayout (this,5) ;

단추들의 이름은 창문이 그리 넓지 않으므로 짧아졌다. 그림 3-8과 그림 3-9를 비교하여보면 두 경우에 간격과 변화가 꼭 같다는것을 알수 있다.

그림 3-9.수평으로 신축되기 전과 후의 수평칸

제8절. 칸에서의 배치

수직칸에 삽입된 창문부품크기는 수평으로 조절할수 없고 창문부품이 그것을 포함하는 칸만큼 넓지 못하면 3가지 선택이 가능하다. 즉 창문부품은 왼쪽, 오른쪽, 또는 중심에 배치할수 있다. 다음 실례는 칸의 꼭대기에 긴 단추를 삽입하고 그 아래에 4개의 다른 고정크기 단추를 삽입함으로써 그림 3-10과 같은 창문을 생성한다.
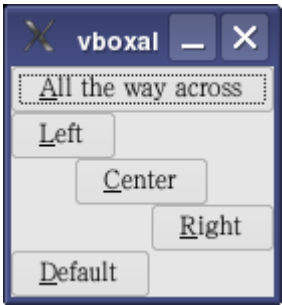
그림 3-10. 왼쪽, 오른쪽, 또는 중심에 배치한 고정크기창문부품

Main

```
1 /* main.cpp */
2 #include <kapplication.h>
3 #include "vboxalign.h"
4
5 int main(int argc, char **argv)
6 {
7     KApplication app(argc, argv, "vboxalign");
8     VBoxAlign *vboxalign = new VBoxAlign();
9     vboxalign->show();
10    app.setMainWidget(vboxalign);
11    return(app.exec());
12 }
```

VBoxAlign머리부파일

```
1 /* vboxalign.h */
2 #ifndef VBOXALIGN_H
3 #define VBOXALIGN_H
4
5 #include <qwidget.h>
6 #include <qlayout.h>
7 #include <qpushbutton.h>
8
9 class VBoxAlign: public QWidget
10 {
11 public:
12    VBoxAlign(QWidget *parent=0,const char *name=0);
13    ~VBoxAlign();
14 private:
15    QPushButton *acrossButton;
16    QPushButton *leftButton;
17    QPushButton *centerButton;
18    QPushButton *rightButton;
19    QPushButton *defaultButton;
20 };
21
22 #endif
```

머리부파일은 클라스들이 창문을 완전히 가로지르는 가변크기단추와 일부분을 차지하는 4개의 고정크기단추들을 비공개성원으로 포함하도록 선언한다.

VBoxAlign

```
1 /* vboxalign.cpp */
2 #include “vboxalign.h”
3
4 VBoxAlign::VBoxAlign(QWidget *parent,const char *name)
5 : QWidget(parent,name)
6 {
7    QVBoxLayout *layout = new QVBoxLayout(this);
8
9    acrossButton = new QPushButton(this);
10   acrossButton->setText("All the way across");
11   acrossButton->setMinimumSize(acrossButton->sizeHint());
12   layout->addWidget(acrossButton);
13
14   leftButton = new QPushButton(this);
15   leftButton->setText("Left");
16   leftButton->setFixedSize(leftButton->sizeHint());
17   layout->addWidget(leftButton,0,AlignLeft);
18
19   centerButton = new QPushButton(this);
20   centerButton->setText("Center");
21   centerButton->setFixedSize(centerButton->sizeHint());
22   layout->addWidget(centerButton,0,AlignCenter);
23
24   rightButton = new QPushButton(this);
25   rightButton->setText("Right");
26   rightButton->setFixedSize(rightButton->sizeHint());
27   layout->addWidget(rightButton,0,AlignRight);
28
29   defaultButton = new QPushButton(this);
30   defaultButton->setText("Default");
31   defaultButton->setFixedSize(defaultButton->sizeHint());
32   layout->addWidget(defaultButton);
```

```
33
34    resize(10,10);
35    layout->activate();
36  }
37  VBoxAlign::~VBoxAlign() { }
```

9～12행은 오직 최소크기만 지정된 단추를 만들고 설치한다. 또한 이 단추의 표식자는 다른것들보다 길고 최소너비는 칸의 다른 단추들보다 크므로 이 단추는 칸자체의 최소너비를 결정한다.

14～17행은 항상 칸의 왼쪽에 놓이는 단추를 만들고 설치한다. 16행의 setFixedSize()호출은 단추의 필수크기를 결정하고 최소한계와 최대한계를 그 크기로 설정한다. 17행의 addWidget()호출은 간격을 0(기정)으로 설정하고 배치를 AlignLeft로 지정한다. 19～22행에서 다른 고정크기단추를 만들고 AlignCenter배치로 칸에 추가한다. 24～27행에서 만든 단추는 AlignRight로 설정된다. 29～32행에서 정의된 마지막 단추에는 구체적인 배치설정을 주지 않았지만 기정값을 AlignCenter로 판명된다.

수평칸은 수직칸과 같은 방법으로 동작하지만 완전히 다른 배치방식이름들을 가지고있다. 수직칸에서 수평으로 맞추는 창문부품들의 이름들은 다음과 같다.

AlignLeft

AlignCenter

AlignRight

마찬가지로 다음것은 수평칸에서 수직으로 배치하는 창문부품들의 3가지 이름들이다.

AlignTop

AlignCenter

ALignBottom

제9절. 배치안의 배치

다음 실례는 어떤 배치를 다른 배치안에 포함하여 그림 3-11에 현시된 창문을 생성하는 방법을 보여준다. 부모배치는 2×2크기의 QGridLayout이다. 그것은 왼쪽웃구석에 QLCDNumber창문부품, 2개의 바닥세포를 차지하는 QSlider창문부품을 포함한다. 오른쪽아래구석의 세포는 QVBoxLayout를 포함하고 그것은 또한 4개 단추들을 포함한다.
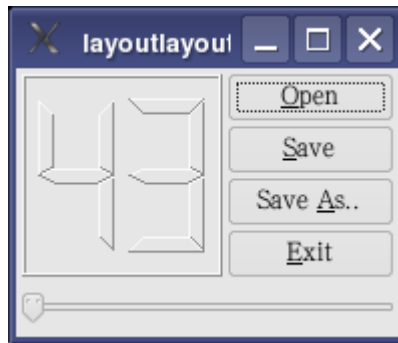
그림 3-11. QGridLayout안에 놓인 QVBoxLayout

알아두기: QGridLayout에 모든 창문부품들을 넣어서 대체로 어떠한 배치도 얻을수 있으며 배치작업을 이와 같이 다시 나누는것이 편리한 때가 있다. 그룹에 의한 창문부품들의 취급은 복잡한 창문프로그람을 짜는 일을 간단화할수 있다.

Main

```
1 /* main.cpp */
2 #include <kapplication.h>
3 #include "layoutlayout.h"
4
5 int main(int argc, char **argv)
6 {
7     KApplication app(argc, argv,  "layoutlayout");
8     LayoutLayout *layoutlayout = new LayoutLayout();
9     layoutlayout->show();
10    app.setMainWidget(layoutlayout);
11    return(app.exec());
12 }
```

LayoutLayout머리부파일

```
1 /* layoutlayout.h */
2 #ifndef LAYOUTLAYOUT_H
3 #define LAYOUTLAYOUT_H
4
5 #include <qwidget.h>
6 #include <qlayout.h>
7 #include <qlcdnumber.h>
8 #include <qpushbutton.h>
9 #include <qslider.h>
```

```
10
11 class LayoutLayout: public QWidget
12 {
13 public:
14     LayoutLayout(QWidget *parent=0,const char *name=0);
15     ~LayoutLayout() { };
16 private:
17     QLCDNumber *lcd;
18     QPushButton *openButton;
19     QPushButton *saveButton;
20     QPushButton *saveasButton;
21     QPushButton *exitButton;
22     QSlider *slider;
23 };
24
25 #endif
```

클라스선언은 현시해야 할 6개 창문부품을 포함한다. 그것들은 비록 다른 배치관리기들에 의해 배치되고 크기가 조종된다 할지라도 LayoutLayout창문부품클라스의 성원들이다. 2개의 계층이 여기에 포함되지만 배치계층은 창문부품계층과 관계없다. 배치계층은 아무일도 하지 않지만 단지 자리표상에서 창문부품의 위치를 지정하고 크기를 계산한다. 그러나 일부 리용에서 매개 창문부품(제일 웃준위창문부품이 기본창문에 사용되는것을 제외하고)은 창문부품나무에서 부모를 가져야 한다.

LayoutLayout

```
1 /* layoutlayout.cpp */
2 #include "layoutlayout.h"
3
4 LayoutLayout::LayoutLayout(QWidget *parent,const char *name)
5 : QWidget(parent,name)
6 {
7     QGridLayout *layout = new QGridLayout(this,2,2,3);
8
9     lcd = new QLCDNumber(this);
10    lcd->setNumDigits(2);
11    lcd->display(43);
12    lcd->setMinimumSize(100,100);
```

```
13    layout->addWidget(lcd,0,0);
14
15    QVBoxLayout *vertButtonLayout = new QVBoxLayout(3);
16    layout->addLayout(vertButtonLayout,0,1);
17
18    openButton = new QPushButton("Open",this);
19    openButton->setMinimumSize(openButton->sizeHint());
20    vertButtonLayout->addWidget(openButton);
21
22    saveButton = new QPushButton("Save",this);
23    saveButton->setMinimumSize(saveButton->sizeHint());
24    vertButtonLayout->addWidget(saveButton);
25
26    saveasButton = new QPushButton("Save As..  ",this);
27    saveasButton->setMinimumSize(saveasButton->sizeHint());
28    vertButtonLayout->addWidget(saveasButton);
29
30    exitButton = new QPushButton("Exit",this);
31    exitButton->setMinimumSize(exitButton->sizeHint());
32    vertButtonLayout->addWidget(exitButton);
33
34    slider = new QSlider(QSlider::Horizontal,this);
35    slider->setMinimumSize(slider->sizeHint());
36    layout->addMultiCellWidget(slider,1,1,0,1);
37
38    resize(10,10);
39    layout->activate();
40 }
```

7행에서 기본배치살창은 2×2살창으로 만들어진다. this인수는 이것이 LayoutLayout객체의 기본배치객체라는것을 지정한다. 또한 살창배치는 포함된 항목들 모두의 주위에 3화소공간을 삽입하도록 만들어진다. 이 창문부품에 의해 현시하려는 모든 항목은 살창배치에 의해 위치지정되여야 한다. (배치에 그것들을 포함하지 않고 자식창문부품들을 현시할수 있지만 창문에서 그것들의 위치는 예측할수 없다.)

9～13행은 2자리수 43을 현시하도록 QLCDNumber객체를 만들어 살창의 왼쪽웃구석에 설치한다.

15행에서 수직배치칸이 만들어진다. 수직칸은 기본배치의 자식으로 포함되므로 지정된 부모가 없다. 그리고 다음 행(16행)에서는 메쏘드 addLayout()를 호출하여 수직칸을 살창배치의 오른쪽웃구석 항목으로서 삽입한다.

이것은 웃끝에 LayoutLayout객체를 가지는 배치계층을 확립하며 7행에서 QGridLayout가 다음 준위에 창조되고 15행에서 QVBoxLayout가 3준위에 만들어진다. 살창을 만드는데 인수 this를 사용하므로 그것을 제일웃준위용기로 설정한다. this인수는 수직칸을 만드는데 쓰이지 않으므로 addLayout()호출이 이루어져 그 계층을 확립한다.

알아두기: QLayout객체는 추가된 자식창문부품 또는 배치를 가지기전에 부모를 가지고있어야 한다. 기본배치는 둘러싸고있는 창문부품의 자식이여야 하며 기타 다른 모든 배치들은 기본배치의 자식(또는 자손)이여야 한다.

18～20행은 QPushButton을 창조하여 수직칸에 추가한다. 단추들에 최소크기가 설정되지만 최대값은 없으므로 단추는 칸이 다 차도록 확장된다. 22～32행은 3개이상의 단추를 만들어 칸에 추가한다. 그 결과 처음에 그림 3-11의 오른쪽웃구석에 보여주는 단추들의 렬이 생긴다.

34～36행은 QSlider를 만들어서 2개 바닥세포를 차지하도록 살창배치에 추가한다. 단추들과 같이 최소크기를 할당하지만 살창배치의 세포들을 채우도록 확장할수 있다.

38행에서 resize()호출은 생성된 창문부품의 처음크기가 최소값이라는것을 담보한다. 최소크기는 LayoutLayout창문부품이 QGridLayout에 요구하여 결정한다. QGridLayout는 QLCDNumber, QSlider 그리고 QVBoxLayout의 최소크기에 경계선간격을 합하여 그 자체의 최소크기를 얻는다. QVBoxLayout는 매 단추들의 최소크기를 얻고 자기의 최소크기를 결정한 다음 최대수평값과 수직값합계를 자기의 최소너비와 최소높이로 사용한다.

요 약

창문부품들의 크기와 위치를 지정하는 여러가지 방법이 있다. 어떤 방법은 자동적으로 창문부품들의 크기를 변경하며 또 어떤 방법은 변경하지 않는다. 마찬가지로 어떤 방법은 창문부품들이 겹치게 하지만 다른 방법은 겹치게 하지 않는다. 자기 응용프로그람에 가장 좋은 방법을 적용하려면 모든 방법을 알고있어야 한다.

· x와 y자리표는 높이와 너비에 따라서 창문부품의 위치와 크기를 고착시키는데 사용될수 있다.

· 가상살창은 기본창문으로 설치될수 있고 창문부품들은 살창에 배치할수 있다.

· 창문부품은 창문에서의 배치에 적합하게 그 크기와 모양을 바꿀수 있다. 이 변화는 창문부품의 최대크기와 최소크기설정에 의해 제한된다.

· 수평칸과 수직칸들은 일직선에 놓인 창문부품들의 행과 렬들을 현시한다. 매개 창문부품들사이에 간격과 위치를 조종할수 있다.

· 수평배치와 수직배치들은 직선상에 놓인 창문부품들의 행과 렬들을 현시한다. 또한

배치는 그 자체가 창문부품이므로 다른 배치, 칸, 또는 살창을 포함하거나 지어는 특정한 x와 y자리표에 의해 배치될수도 있다.

이 장은 제일 웃준위창문을 만드는 방법을 설명하였다. 다음 장은 튀여나오기창문을 만드는 방법을 설명한다. 제일 웃준위창문용 창문부품들의 위치를 지정하는 모든 도구와 방법은 튀여나오기창문안에 있는 창문부품들의 위치지정에도 사용된다. 기본차이는 대화칸이 사용자에게 정보를 현시하고 몇가지 종류의 응답을 돌려주는데 사용되는 일시창문이라는것이다.

제4장. 튀여나오기대화칸의 현시

학습내용

- 간단한 대화칸의 창조와 현시
- 대화칸으로부터 신호를 받아들이는 처리부의 실현
- 대화칸에서 처리부들에 전송할 신호의 창조
- 사용자정의대화칸 만들기
- KDialogBase의 확장
- KMessageBox에 정의된 일반대화칸의 펼치기

대화칸은 보통 일시적인 창문으로서 사용자에게 특정한 정보를 현시하고 사용자로부터 특정한 정보를 요구한다. 많은 대화칸은 매우 간단하고 오직 예 또는 아니라는 대답만 요구하지만 매우 복잡한 대화칸은 흔하지 않고 정보를 현시하고 받아들이는 창문부품들의 여러 페지를 포함한다.

대화칸은 응용프로그람창문을 부모로 하지만 항상 독립창문으로 현시기에 나타난다. 그것은 일부 창문조종과 차림표가 없는것을 제외하면 제일 웃준위창문과 거의 같아보인다.

확장가능한 대화칸기초클라스가 많으므로 프로그람이 대화칸을 만드는 방법도 많다. 또한 기초클라스자체는 상대적으로 간단한 대화칸을 만드는데 사용될수 있다. 이 장은 제각기 자기의 우점을 가지고있는 기초클라스들로부터 대화칸을 만드는 여러가지 방법을 실례를 들어 설명한다.

제1절. 간단한 대화칸

QDialog창문부품은 대화칸을 만드는데 사용할수 있는 기초클라스이지만 또한 간단한 창문부품들의 배치를 조종하는데 직접 사용될수 있다. QDialog는 빈 창문만 현시하는 간단한 창문부품이지만 창문부품들의 용기로서 사용될수 있고 독립창문에 그것을 현시할 능력을 가지고있다. 또한 창문에 단추들을 추가하여 거기에 응답하기 위한 기본기능들이 있다.

다음 실례는 대화칸을 만들고 현시하는 방법을 보여준다. 그것은 QDialog객체를 대화칸으로 현시하는 방법을 보여준다. 그림 4-1의 왼쪽그림이 제일 웃준위창문이다. 그것은 오른쪽에 있는 대화칸을 펼치는데 사용되는 하나의 단추를 포함한다. 대화칸에는 그것을 완료하는데 사용할수 있는 단추가 있다.

그림 4-1. 제일 웃준위창문과 그것이 펼치는 대화칸

Main

```
1 /* simple.cpp */
2 #include <kapplication.h>
3 #include "toplevel.h"
4
5 int main(int argc, char **argv)
6 {
7     KApplication app(argc, argv,  "simple");
8     TopLevel toplevel;
9     toplevel.show();
10    app.setMainWidget(&toplevel);
11    return(app.exec());
12 }
```

프로그람의 기본함수는 KApplication객체를 만들고 TopLevel창문부품을 그 제일 웃준위창문으로 만들고 설치한다.

TopLevel머리부파일

```
1 /* toplevel.h */
2 #ifndef TOPLEVEL_H
3 #define TOPLEVEL_H
4
5 #include <qwidget.h>
6
7 class TopLevel: public QWidget
8 {
9     Q_OBJECT
10 public:
11    TopLevel(QWidget *parent=0,const char *name=0);
12 private slots:
13    void popupDialog();
14 };
15
16 #endif
```

TopLevel창문은 QWidget를 계승하므로 창문부품이며 프로그람의 제일 웃준위창문용으로 설계된다. 클라스선언은 오직 구성자와 단추에 응답하는데 필요한 최소량의 정보를 포함한다. 9행의 Q_OBJECT마크로는 12행과 13행에서 선언한 처리부를 위해 제시되여야 한다. 처

리부메쏘드 popupDialog()는 단추를 누를 때마다 호출된다.

TopLevel

```
1 /* toplevel.cpp */
2 #include "toplevel.h"
3 #include <qdialog.h>
4 #include <qpushbutton.h>
5
6 TopLevel::TopLevel(QWidget *parent,const char *name)
7       : QWidget(parent,name)
8 {
9       setMinimumSize(200,80);
10      setMaximumSize(200,80);
11
12      QPushButton *button = new QPushButton("Pop Up",this);
13      button->setGeometry(50,20,100,40);
14      connect(button,SIGNAL(clicked()),
15              this,SLOT(popupDialog()));
16 }
17 void TopLevel::popupDialog()
18 {
19      QDialog *dialog = new QDialog(0,  "popup",FALSE);
20      dialog->setCaption("A QDialog Window");
21      dialog->setMinimumSize(200,80);
22      dialog->setMaximumSize(200,80);
23
24      QPushButton *button =
25              new QPushButton("Pop Down",dialog);
26      button->setGeometry(50,20,100,40);
27      connect(button,SIGNAL(clicked()),
28              dialog,SLOT(accept()));
29
30      dialog->show();
31 }
```

이 코드는 대화칸을 만들고 현시하는 메쏘드를 비롯한 TopLevel창문부품의 정의를 포함한다.

TopLevel창문부품은 프로그람의 기본창문이다. 9,10행의 구성자에서 창문크기를 변경할수 없게 설정되였으며 최대크기와 최소크기는 같은 값으로 설정된다. 12행에서 단추가 만들어진다. 13행의 기하학적크기설정은 단추의 높이와 너비 그리고 그 왼쪽웃구석의 위치를 지정한다. 14행에서 connect()호출은 단추의 clicked()신호로부터 발생하는 사건들이 TopLevel창문부품의 popupDialog()로 넘어간다는것을 의미한다.

17행의 메쏘드 popupDialog()는 TopLevel창문에서 사용자가 단추를 찰칵할 때마다 호출된다. 19～22행은 QDialog창문부품의 실례를 만들고 그 크기와 제목을 지정한다. 24～26행에서 창문부품이 보관하려는 단추를 만들고 크기를 지정한다. 27행에서 connect()호출은 단추의 clicked()로부터 발생하는 신호가 QDialog창문부품안의 accept()메쏘드로 넘어가도록 한다. 30행에서 show()호출은 QDialog창문부품이 자기 창문에 현시되게 한다.

QDialog창문부품은 사용자가 그것을 리용하여 모든 대화칸을 창조할수 있도록 충분히 유연하다. 이 실례에서처럼 처리부메쏘드 accept()에 의해 대화칸을 닫을수 있고 또한 reject()를 호출하여 대화칸을 닫을수 있다. 두 메쏘드들사이의 유일한 차이는 그것들중 하나가 결과를 TRUE로 설정하고 다른것은 FALSE로 설정하는것이다. 이 설정값은 일반적으로 대화칸에 나타나는 Cancel과 OK단추에 대응된다.

대화칸이 accept() 혹은 reject()호출에 의하여 닫길 때 그것은 파괴되지 않는다. 그 창문은 hide()메쏘드호출에 의해서 닫긴다. 이것은 프로그람이 설정값을 읽을수 있는 우점을 가지고 있으나 자체로 대화칸을 제거해야 한다. 그러나 같은 대화칸을 반복 사용하려면 그것을 일단 만들고 필요할 때 hide()와 show()를 호출할수 있다.

이 실례는 2가지 문제점이 있다. 첫째로, 요구되는만큼 대화칸을 많이 펼칠수 있다. 즉 Pop Up단추를 찰칵할 때마다 새로운 대화칸이 생겨나고 실행된다. 둘째로, Push Down단추로 대화칸을 닫을 때 그것은 삭제되지 않는다. 그것은 accept()호출에 의해 닫기지만 여전히 존재하며 프로그람은 그 지적자를 가지지 않는다.

알아두기: 창문제목을 설정하고 크기와 여백을 변경하는 메쏘드들이 추가된것외에 QDialog창문부품과 거의 같은 KDialog창문부품이 있다. QDialog을 사용할수 있는곳에서는 어디서나 KDialog를 사용할수 있다.

제2절. 신호와 처리부의 사용

이 실례는 QDialog를 기초클라스로 사용하여 문자렬을 받아들이는 대화칸을 만든다. 그리고 OK 또는 Apply단추가 선택되면 문자렬은 프로그람의 기본창문에 전송되고 제목띠를 새로운 표제로 설치한다. 그림 4-2의 왼쪽창문은 기본창문과 그 한개 단추를 보여준다. 오른쪽에는 새 제목문자렬을 받아들이고 표시하는 대화칸이다.
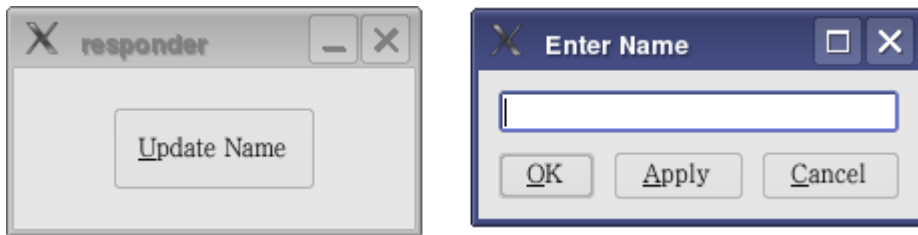
그림 4-2. 단추와 그것이 펼치는 대화칸

Mainline

```
1 /* responder.cpp */
2 #include <kapplication.h>
3 #include "mainwidget.h"
4
5 int main(int argc, char **argv)
6 {
7     KApplication app(argc, argv,  "responder");
8     MainWidget mainwidget;
9     mainwidget.show();
10    app.setMainWidget(&mainwidget);
11    return(app.exec());
12 }
```

프로그람의 기본함수는 아주 간단하다. 8행과 9행에서 MainWidget객체를 만들고 10행은 그것을 기본창문으로 설치한다.

MainWidget머리부파일

```
1 /* mainwidget.h */
2 #ifndef MAINWIDGET_H
3 #define MAINWIDGET_H
4
5 #include <qwidget.h>
6 #include <qstring.h>
7
8 class MainWidget: public QWidget
9 {
10    Q_OBJECT
11 public:
12    MainWidget(QWidget *parent=0,const char *name=0);
13 private slots:
```

```
14    void popupEnterName();
15    void changeCaption(QString &);
16 };
17
18 #endif
```

이 실례에서 기본창문으로 작용하는 창문부품의 머리부파일이다. 이 클라스는 12행에서 선언된 구성자외에 두개의 처리부를 포함한다. popupEnterName()처리부는 대화칸을 펼치고 changeCaption()처리부는 이 창문부품의 제목(기본창문의 제목)본문을 변경한다.

이 클라스에 처리부들이 있으므로 클라스의 첫 성원으로서 Q_OBJECT마크로를 사용한다. Q_OBJECT안의 정의들은 이 머리부파일을 표준C++코드로 콤파일하게 하며 그것은 처리부와 신호를 조종하는데 필요한 코드를 생성하기 위하여 메타객체콤파일러(Meta Object Compiler, MOC)에서 사용하는 특별한 정보를 삽입한다.

MainWidget

```
1 /* mainwidget.cpp */
2 #include "mainwidget.h"
3 #include "entername.h"
4 #include <qpushbutton.h>
5
6 MainWidget::MainWidget(QWidget *parent,const char *name)
7     : QWidget(parent,name)
8 {
9     setMinimumSize(200,80);
10    setMaximumSize(200,80);
11
12    QPushButton *button =
13            new QPushButton("Update Name",this);
14    button->setGeometry(50,20,100,40);
15    connect(button,SIGNAL(clicked()),
16            this,SLOT(popupEnterName()));
17 }
18 void MainWidget::popupEnterName()
19 {
20    EnterName *dialog = new EnterName(0,  "entername");
21    connect(dialog,SIGNAL(captionString(QString &)),
22            this,SLOT(changeCaption(QString &)));
```

```
23    dialog->exec();
24    delete dialog;
25 }
26 void MainWidget::changeCaption(QString &caption)
27 {
28    setCaption(caption);
29 }
```

이 클라스는 응용프로그람의 기본창문으로 사용된다. 그것은 이 절의 앞에서 보여준 그림 4-2의 왼쪽에 창문으로 현시된다.

이 클라스는 7행에서 QWidget를 계승하므로 창문부품이다. 9행과 10행은 최대크기와 최소크기를 같은 값으로 설정하여 이것을 고정크기창문부품으로 만든다.

QPushButton은 12행과 13행에서 창조되고 14행에서 창문중심에 배치된다. 단추는 마우스에 눌리울 때마다 발생하는 clicked()라는 신호를 가지고있다. 15행의 connect()호출은 clicked()신호가 발생할 때마다 국부처리부메쏘드 popupEnterName()가 호출되도록 설정한다.

알아두기: 메쏘드를 처리부로 리용하여도 그것을 직접 호출할수 있다. 이 실례에서 메쏘드 popupEnterName()는 신호에 의해 호출되지만 이 클라스의 다른 메쏘드로부터 지어 다른 클라스로부터도 호출될수 있다. 처리부는 신호를 포착하는데 사용할수 있는 추가특성을 가지는 보통 메쏘드이다.

18행의 메쏘드 popupEnterName()는 EnterName대화칸을 만들고 사용자에게 새로운 제목을 묻는다. 21행에서 connect()호출은 대화칸에서 captionString()신호가 changeCaption()국부처리부호출을 만들도록 련결한다.

23행의 exec()호출은 대화칸을 펼치고 대화칸은 입력대기렬에로의 호출을 독점한다. 이 메쏘드는 사용자가 OK 또는 Cancel단추를 선택하여 응답할 때까지 되돌아가지 않는다. 사용자가 응답할 때까지 이 응용프로그람이 소유한 다른 창문은 마우스 또는 건반신호들을 받지 않는다. 23행에서 응답이 이루어진 후에 대화칸이 삭제된다.

26행의 처리부메쏘드는 오직 사용자가 대화칸에서 OK단추를 선택할 때만 호출되므로 새로운 제목문자렬이 기본창문용으로 설정된다.

EnterName머리부파일

```
1 /* entername.h */
2 #ifndef ENTERNAME_H
3 #define ENTERNAME_H
4
5 #include <qdialog.h>
6 #include <qlineedit.h>
7 #include <qpushbutton.h>
```

```
8
9 class EnterName: public QDialog
10 {
11     Q_OBJECT
12 private:
13     QLineEdit *lineedit;
14     QPushButton *okButton;
15     QPushButton *applyButton;
16     QPushButton *cancelButton;
17 public:
18     EnterName(QWidget *parent=0,const char *name=0);
19 private slots:
20     void okButtonSlot();
21     void applyButtonSlot();
22     void cancelButtonSlot();
23 signals:
24     void captionString(QString &);
25 };
26
27 #endif
```

이 머리부파일은 새 제목문자렬을 묻는데 사용되는 튀여나오기대화칸의 클라스를 선언한다. 그것은 단추찰칵을 받아들이는 처리부들과 새 제목본문과 함께 보내는 신호를 가지고있다.

이 클라스는 9행에서 기초클라스로서 QDialog를 사용하므로 대화칸선언이다. 처리부나 신호를 포함하는 클라스는 반드시 첫 성원으로서 Q_OBJECT마크로를 포함하여야 한다. 13～16행은 대화칸의 성원들을 구축하는데 사용할 4개 창문부품용 기억공간을 선언한다.

19～22행은 처리부들의 이름을 지정한다. okButtonSlot(), applyButtonSlot() 및 cancelButtonSlot()메쏘드들은 단추찰칵을 받아들이기 위한 국부처리부들이다. 24행의 신호 captionString()은 사용자가 새 제목문자렬을 내보낼 때마다 발생되는 신호이다.

EnterName

```
1 /* entername.cpp */
2 #include "entername.h"
3 #include <qdialog.h>
4 #include <qlayout.h>
5
6 EnterName::EnterName(QWidget *parent,const char *name)
```

```
7      : QDialog(parent,name,TRUE)
8 {
9      QString caption("Enter Name");
10     setCaption(caption);
11
12     QVBoxLayout *vLayout = new QVBoxLayout(this,10);
13
14     lineedit = new QLineEdit(this);
15     vLayout->addWidget(lineedit);
16
17     QHBoxLayout *hLayout = new QHBoxLayout(vLayout,10);
18
19     okButton = new QPushButton("OK",this);
20     connect(okButton,SIGNAL(clicked()),
21             this,SLOT(okButtonSlot()));
22     hLayout->addWidget(okButton);
23
24     applyButton = new QPushButton("Apply",this);
25     connect(applyButton,SIGNAL(clicked()),
26             this,SLOT(applyButtonSlot()));
27     hLayout->addWidget(applyButton);
28
29     cancelButton = new QPushButton("Cancel",this);
30     connect(cancelButton,SIGNAL(clicked()),
31             this,SLOT(cancelButtonSlot()));
32     hLayout->addWidget(cancelButton);
33 }
34 void EnterName::okButtonSlot()
35 {
36     QString str = lineedit->text();
37     emit captionString(str);
38     accept();
39 }
40 void EnterName::applyButtonSlot()
41 {
```

```
42    QString str = lineedit->text();
43    emit captionString(str);
44 }
45 void EnterName::cancelButtonSlot()
46 {
47    reject();
48 }
```

이 클라스는 사용자가 본문을 입력하고 적당한 단추를 선택함으로써 본문이 기본창문의 제목으로 설치되게 하는 대화칸이다.

6행에서 EnterName구성자에로의 인수가 7행에서 QDialog기초클라스에로 넘어간다. QDialog의 셋째 인수는 TRUE이고 이것이 이행금지대화칸이라는것을 지정한다.

12행에서 창조한 수직칸은 창문을 위한 기본용기로 사용된다. 14행에서 창조한 QLineEdit객체는 수직칸의 꼭대기에 삽입된다. 수평칸은 수직칸의 자식으로 만들어지는데 그것은 수평칸이 수직칸(QLineEdit창문부품의 바로 아래)의 다음 성원이 되게 한다. 수평칸에 3개 단추를 삽입하여 (22, 27 그리고 32행) 이전에 그림 4-2에서 오른쪽에 보여준 배치를 완성한다.

20, 25 그리고 30행에서 connect()메쏘드호출은 단추들의 clicked()신호를 매개 처리부에 련결한다.

34행의 처리부메쏘드 okButtonSlot()는 OK단추를 찰칵할 때마다 호출된다. QLineEdit객체의 text()호출은 사용자가 입력한 문자렬을 받아들인다. 37행은 captionString()이라는 신호를 발생한다. 신호는 메쏘드를 호출하는데 사용하는것과 거의 같은 문법으로 발생되지만 그것이 메쏘드가 아니라 내보내는 신호라는것을 의미하는 예약어 emit를 앞에 붙인다. 처리부메쏘드는 38행에서 accept()를 호출함으로써 완료한다. 이 호출은 내부기발을 TRUE로 설정하여 사용자로부터 긍정적인 응답이 있다는것을 지적하고 hide()를 호출하여 창문부품을 보이지 않게 한다.

Apply단추를 찰칵할 때마다 40행의 applyButtonSlot()가 호출된다. OK단추처리부로 수행한것처럼 신호메쏘드 captionString()을 사용하여 문자렬을 얻고 신호를 발생시킨다. accept()메쏘드는 대화칸이 보임상태로 되여있으므로 호출되지 않는다.

Cancel단추를 찰칵할 때마다 45행의 cancelButtonSlot()가 호출된다. 사용자가 제목이름을 변경하는 작용을 취소하였으므로 어떤 신호도 발생하지 않는다. reject()호출이 이루어지고 내부기발이 FALSE로 설정되고 대화칸의 창문이 닫긴다.

Makefile

```
1 INCL= -I$(QTDIR)/include -I$(KDEDIR)/include
2 CFLAGS= -O2 -fno-strength-reduce
3 LFLAGS= -L$(QTDIR)/lib -L$(KDEDIR)/lib -L/usr/X11R6/lib
4 LIBS= -lkdecore -lkdeui -lqt -lX11 -lXext -ldl
```

```
5 CC=g++
6
7 recaption: recaption.o mainwidget.o moc_mainwidget.o \
8 entername.o moc_entername.o
9 $(CC) $(LFLAGS) -o recaption recaption.o \
10 mainwidget.o moc_mainwidget.o \
11 entername.o moc_entername.o $(LIBS)
12
13 recaption.o: recaption.cpp mainwidget.h
14 mainwidget.o: mainwidget.cpp mainwidget.h
15 moc_mainwidget.cpp: mainwidget.h
16 $(QTDIR)/bin/moc mainwidget.h -o moc_mainwidget.cpp
17 entername.o: entername.cpp entername.h
18 moc_entername.cpp: entername.h
19 $(QTDIR)/bin/moc entername.h -o moc_entername.cpp
20
21 clean:
22 rm -f recaption
23 rm -f *.o
24 rm -f moc_*
25
26 .SUFFIXES: .cpp
27
28 .cpp.o:
29 $(CC) -c $(CFLAGS) $(INCL) -o $@ $<
```

이 코드가 설명하는것처럼 처리부 또는 신호가 원천에 포함될 때 특별한 항목들이 makefile에 포함되여야 한다. 코드는 직접 콤파일될뿐아니라 또한 MOC콤파일러에 의해 콤파일될 독립적인 원천파일로 변환된다.

7행에는 recaption을 련결하는 의존관계목록가 있다. 거기에는 .cpp파일들의 이름과 일치하는 .o파일들뿐아니라 4개 기호 moc_로 시작하는 .o파일들도 있다. 첫 성원으로 Q_OBJECT를 포함하는 클라스(처리부 또는 신호를 가지는 클라스)는 MOC콤파일러에 의해 처리된 머리부파일을 가지고 있어야 한다. 15행의 의존관계는 원천파일 moc_mainwidget.cpp가 원천파일 mainwidget.h에 의존하도록 설정한다. 16행의 지령은 입력으로서 mainwidget.h를 사용하여 moc_mainwidget.cpp를 만든다. 그 다음에 moc_mainwidget.cpp는 moc_mainwidget.o로 콤파일되여 9행에서 련결에 포함된다.

제3절. 신호와 처리부검사목록

신호와 처리부는 아주 간단히 만들수 있다. 대부분의 작업은 마크로와 MOC콤파일러형식으로 자동화된다. 신호를 발송하는 과정은 처리부가 신호를 받아들이는 과정과 완전히 독립된다. 객체는 신호를 받아들이는 처리부들이 얼마인지 모르고도 임의의 개수의 신호를 낼수 있다. 다음에 신호를 만들어서 처리부들에 보내기 위하여 수행해야 할 모든 단계를 포함한다.

① 클라스선언의 첫 행으로서 Q_OBJECT마크로를 추가한다. 클라스의 다른 항목들은 반두점으로 끝나야 하지만 Q_OBJECT마크로는 그렇지 않다. 그러나 만일 좋아한다면(콤파일러가 간단히 반두점을 버리므로) 포함할수 있다. 실례로 Receiver라는 클라스의 선언은 이 방법으로 시작한다.

```
class Sender {
    Q_OBJECT
    ...
```

임의의 개수의 처리부와 신호를 객체에서 정의할수 있으나 Q_OBJECT마크로는 오직 한번 포함해야 한다.

② 클라스선언에 신호의 원형을 추가한다. 실례로 신호가 인수로서 문자렬객체를 보내려고 한다면 원형은 다음과 같다.

```
…
signals:
    void  newName(QString  &name);
    …
```

실제 메쏘드는 없으므로 공개나 비공개지정도 없다. 이것은 받아들이는 처리부를 호출하는데 쓰이는 원형의 선언이다.

③ emit명령문을 사용하여 신호를 받아들이는 모든 처리부메쏘드를 호출한다. 이것은 emit예약어가 호출앞에 놓이는것을 제외하면 국부메쏘드호출과 같은 문법으로 수행된다.

```
QString  name;
emit  newName(name);
```

신호메쏘드본체의 실제정의는 없다. emit지령은 국부메쏘드를 찾지 않고 그대신에 이 신호에 련결되여있는 처리부목록안의 모든 처리부메쏘드를 호출한다.

다음 단계는 처리부를 만들어서 신호에 련결하는데 필요하다.

① 신호에서와 같이 처리부는 Q_OBJECT마크로를 클라스선언의 선두에 포함해야한다.

```
class Receiver {
    Q_OBJECT
    …
```

② 클라스선언에 처리부메쏘드들의 원형을 추가한다. 원형은 그것이 받아들일 신호와 같아야 한다. 즉 같은 인수모임을 가져야 한다. 처리부는 메쏘드이고 처리부로 사용될뿐아니라 직접 호출될수 있으므로 처리부메쏘드를 공개적으로 사용할수 있다.

```
…
public  slots:
    void  nameChange(QString  &name);
```

오직 신호들을 받아들일 목적으로 사용되는 처리부는 비공개로 선언한다.

③ 신호를 발생시키는 클라스를 선언하는 머리부파일을 포함한다.

④ 그 신호를 발생시키는 클라스의 실례를 창조하는 코드를 쓴다. 그것은 신호에 처리부를 련결하기 위하여 필요하다.

⑤ 신호에 처리부를 련결한다. 이것은 흔히 구성자에서 수행되지만 객체가 후에 만들어지면 후에 수행될수 있다. connect()메쏘드호출은 특정한 신호가 발생될 때마다 호출되는 메쏘드들의 목록에 처리부를 추가한다. connect()호출은 다음과 같다.

```
connect(sender, SIGNAL(newName(QString &), this, SLOT(nameChange(QString &)) );
```

처음 2개 인수는 신호의 원천을 지정하고 두번째 2개는 목적처리부를 지정한다. 마크로 SIGNAL()과 SLOT()는 모두 완전한 메쏘드원형을 요구하며 원형은 두 메쏘드들의 인수모임은 서로 같아야 한다.

emit문이 신호를 보낼 때 마치도 자기 프로그람이 직접 처리부메쏘드중 하나를 호출한 것처럼 정확하다. 즉 프로그람은 처리부메쏘드가 되돌아갈 때까지 계속할수 없다. 그러므로 보통 신호발생기가 정지하지 않도록 처리부메쏘드내에서의 처리를 될수록 간단하게 해야 한다. 신호의 발생기는 사용자대면부프로쎄스일수 있고 느리거나 완만한 조작을 출현시킬수 있다.

순환이 일어나지 않도록 매우 주의해야 한다. 처리부메쏘드가 직접이든 간접이든 원천의 처리부에 의해 받아들인 신호를 발생시키는 메쏘드를 실행하면 신호들은 계속 처리부들을 호출하며 프로그람은 중단된다. 실례로 first()라는 메쏘드가 신호 A를 발생시키고 신호 A를 second()처리부가 받아들이고 처리부 second()는 신호 B를 발생시키고 first()처리부는 신호 B를 받아들이면 순환이 존재하고 그것은 프로그람이 중단할 때까지(사용자가 기다리기 실증날 때까지) 계속된다.

또한 connect문의 처리부와 신호메쏘드들이 일치하는 인수를 가지지 않으면 프로그람이 실행되고 있을 때 참고를 해결하려는 시도가 이루어질 때까지 오유통보문을 얻지 못한다.

이것을 피하기 위하여 처리부와 신호에 만드는 모든 추가 혹은 변경을 정확히 검사하여야 한다. 유일한 오유통보문은 connect()메쏘드가 쌍을 찾는데 실패하고 그후에 프로그람이 조용히 신호들을 무시할 때 표준말단(표준출력)에 써내보내는 문자렬이다. 그리고 오직 지령행으로부터 응용프로그람을 실행할 때 콘솔출력을 볼수 있다.

제4절. KDialogBase

창문부품 KDialogBase는 대화칸의 한 종류이다. 대부분의 대화칸은 같은 기본형식 즉 바닥에 단추들의 행을 가지는 자료항목창문부품들의 집합이다. 그와 함께 KDialogBase창문부품은 단추들의 기본(짜넣은)행으로 설계되였다. 다음 실례프로그람은 그림 4-3과 같이 KDialogBase의 기정환경구성을 보여준다.
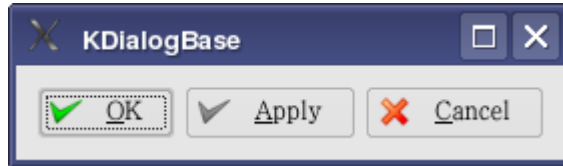
그림 4-3. KDialogBase창문의 기정단추

Mainline

```
1 /* kdbsimple.cpp */
2 #include <kapplication.h>
3 #include "mainwidget.h"
4
5 int main(int argc, char **argv)
6 {
7     KApplication app(argc, argv,  "kdbsimple");
8     MainWidget mainwidget;
9     mainwidget.show();
10    app.setMainWidget(&mainwidget);
11    return(app.exec());
12 }
```

8행에서 만들어진 mainwidget는 10행에서 응용프로그람의 기본창문으로 설정된다.

MainWidget머리부파일

```
1 /* mainwidget.h */
2 #ifndef MAINWIDGET_H
3 #define MAINWIDGET_H
4
5 #include <qwidget.h>
6
7 class MainWidget: public QWidget
8 {
9     Q_OBJECT
```

```
10 public:
11     MainWidget(QWidget *parent=0,const char *name=0);
12 private slots:
13     void popupKdb();
14 };
15
16 #endif
```

기본창문은 오직 2개의 메쏘드를 가지고있다. 하나는 구성자이고 다른것은 누름단추에 련결될 처리부이다. popupKdb()의 목적은 KDialogBase창문을 현시하는것이다.

MainWidget

```
1 /* mainwidget.cpp */
2 #include "mainwidget.h"
3 #include <qpushbutton.h>
4 #include <kdialogbase.h>
5
6 MainWidget::MainWidget(QWidget *parent,const char *name)
7     : QWidget(parent,name)
8 {
9     setMinimumSize(200,80);
10    setMaximumSize(200,80);
11
12    QPushButton *button =
13            new QPushButton("Popup",this);
14    button->setGeometry(50,20,100,40);
15    connect(button,SIGNAL(clicked()),
16            this,SLOT(popupKdb()));
17 }
18
19 void MainWidget::popupKdb()
20 {
21    KDialogBase *dialog = new KDialogBase(this,
22            "kdbwidget",TRUE);
23    dialog->exec();
24    delete dialog;
25 }
```

이 창문부품은 실례의 기본창문으로 사용된다. 그것은 오직 "Popup"단추와 단추를 찰칵할 때마다 실행하는 처리부를 포함한다.

단추를 찰칵할 때마다 KDialogBase창문부품이 구성된다(21행). 23행에서 exec()호출은 그림 4-3과 같은 대화칸을 펼친다. 창문이 처음으로 나타날 때 OK단추가 선택되므로 간단히 Return 또는 Enter건을 누르는것은 OK를 찰칵하는것과 같다. 또한 그림으로부터 알수 있는것처럼 매개 단추는 지정된 지름건을 가진다. 실례로 Alt+C를 누르는것은 Cancel단추를 선택하는것과 같다.

Apply단추에 처리부를 련결하지 않으면 그것은 아무 일도 하지 않는다. Cancel과 OK단추는 모두 대화칸을 완료한다. 단추들을 사용하기 위하여 간단히 OK와 Apply단추들을 대화칸으로부터 자료를 받아들이고 처리하는 처리부에 련결한다.

제5절. KDialogBase단추

앞의 실례는 3개의 기정단추가 OK, Apply 그리고 Cancel이라는것을 보여주었다. 그러나 다른 단추들도 있으며 자체의 3개 단추를 추가할수 있다. 다음 실례는 그림 4-4와 같이 8개의 단추를 모두 보여주는 창문을 현시한다.

그림 4-4. KDialogBase클라스의 단추선택

머리부파일과 프로그람의 기본함수는 이전 실례들과 같다. 프로그람들사이의 유일한 차이는 KDialogBase의 구성자에 넘기는 인수모임이라는것이다.

MainWidget

```
1 /* mainwidget.cpp */
2 #include "mainwidget.h"
3 #include <qpushbutton.h>
4 #include <kdialogbase.h>
5
6 MainWidget::MainWidget(QWidget *parent,const char *name)
7       : QWidget(parent,name)
8 {
9       setMinimumSize(200,80);
10      setMaximumSize(200,80);
11
12      QPushButton *button =
13              new QPushButton("Popup",this);
```

```
14    button->setGeometry(50,20,100,40);
15    connect(button,SIGNAL(clicked()),
16            this,SLOT(popupKdb()));
17 }
18
19 void MainWidget::popupKdb()
20 {
21    QString caption("All Buttons");
22    QString button1("User1");
23    QString button2("User2");
24    QString button3("User3");
25
26    int buttons = KDialogBase::Ok
27            | KDialogBase::Apply
28            | KDialogBase::Cancel
29            | KDialogBase::Help
30            | KDialogBase::Default
31            | KDialogBase::User1
32            | KDialogBase::User2
33            | KDialogBase::User3;
34
35    KDialogBase *dialog = new KDialogBase(
36            this, // 부모
37            "kdbwidget", // 이름
38            TRUE, // 이행금지
39            caption, // 제목
40            buttons, // 단추마스크
41            KDialogBase::Cancel, // 기정단추
42            FALSE, // 분리선
43            button1, // 단추제목
44            button2, // 단추제목
45            button3); // 단추제목
46    dialog->exec();
47    delete dialog;
48 }
```

이 창문부품은 응용프로그람의 기본창문으로 사용된다. 그것은 찰칵할 때 KDialogBase 창문을 현시하는 단추를 포함한다.

처리부메쏘드 popupKdb()는 12행에서 창조한 기본창문단추를 찰칵할 때마다 실행된다. KDialogBase창문부품은 35행의 구성자에 의해 만들어진다. 이전의 실례에서 보여준것처럼 모든 인수는 정의된 기정값을 가지지만 이 실례는 그 매개의 값을 지정한다. 지정된 값들을 표 4-1에서 설명한다.

표 4-1. KDialogBase의 구성자가 받아들이는 파라메터들

파라메터	설명
부모	부모창문부품인데 이것은 보통 KDialogBase를 펼치는 창문부품이다. 기정값은 NULL이다.
이름	창문부품의 내부이름인데 내부목적과 오유통보문을 발생시키는데 사용된다. 기정값은 NULL이다.
이행금지	TRUE로 설정되면 이 창문부품은 이행금지로서 현시한다. FALSE로 설정되면 이행허용으로 된다. 기정값은 TRUE이다.
제목	창문꼭대기의 제목띠에 있는 제목의 본문. 기정값은 응용프로그람의 이름이다.
단추마스크	대화칸에서 능동으로 되여야 할 단추들을 지정하는 1bit기발들의 모임. 기정값은 3개의 단추 즉 Ok, Apply 그리고 Cancel이다.
기정단추	대화칸이 처음으로 나타날 때 선택되여야 할 단추(이 단추는 Return 혹은 Enter건에 응답한다). 기정값은 Ok단추이다.
분리선	TRUE이면 단추우에 분리선이 생겨난다. FALSE이면 분리선이 없다. 기정값은 FALSE이다.
단추제목	이것은 사용자정의단추에 나타나는 본문이다. 기정값은 NULL로서 사용자단추를 비게 한다.

그림 4-4에서 보여준것처럼 단추들의 순서는 KDialogBase창문부품에 의하여 내적으로 결정한다. 포함할 단추들을 결정할수 있지만 그것들의 출현순서는 늘 보여주는 순서이다.

표 4-2는 KDialogBase창문부품에서 사용할수 있는 모든 단추들을 보여준다. 이 단추들로부터 정보를 받는것은 적합한 KDialogBase신호메쏘드에 자기 처리부메쏘드를 련결하는 문제이다. 대화칸을 닫는 단추들은 결과를 가리키는 상태코드를 설정한다. 결과코드를 얻으려면 여기에 보여준것처럼 앞 실례에서 46행과 47행사이에 행들을 삽입하여야 한다.

```
dialog->exec();
int resultCode = dialog->result();
delete dialog;
```

표 4-2. KDialogBase의 단추와 신호

단추	신호	설명
Apply	applyClicked()	Apply단추와 Try단추가 모두 지정되면 Try단추는 나타나지 않는다.
Cancel	closeClicked()	Close단추대신에 사용할수 있다.
Close	closeClicked()	Close와 Cancel가 모두 지정되면 오직 Close만 나타난다. 결과코드는 FALSE로 설정되고 대화칸은 닫긴다.
Default	defaultClicked()	
Help	helpClicked()	메쏘드 invokeHTMLHelp()를 호출하여 setHelp()호출에 의해 정의된 도움본문을 현시한다.
No	noClicked()	대화칸이 통보칸방식일 때 User1단추위치에 나타난다. 결과코드는 FALSE로 설정되고 대화칸은 닫긴다.
OK	okClicked()	결과코드는 TRUE로 설정되고 대화칸은 닫긴다.
Try	tryClicked()	Apply단추대신에 사용할수 있다.
User1	user1Clicked()	구성자에서 인수는 표식자를 지정한다. 이 단추는 통보칸방식에서 No단추로 교체된다.
User2	user2Clicked()	구성자에서 인수는 표식자를 지정한다. 이 단추는 통보칸방식에서 Yes단추로 교체된다.
User3	user3Clicked()	
Yes	None	이 단추는 대화칸이 통보칸방식일 때 User2단추의 장소에 나타난다. 결과코드는 TRUE로 설정되고 대화칸은 닫긴다.

제6절. KDialogBase를 리용한 대화칸의 만들기

다음 실례프로그람은 대화칸의 기초클라스로서 KDialogBase를 사용하여 사용자가 본문행과 2개의 옹근수값을 지정할수 있게 한다. 이 실례에서 대화칸에 입력한 정보는 표식자에의해 현시된 본문과 기본창문크기를 변경하는데 사용된다. 그림 4-5는 대화칸(오른쪽)에 보여준 값들에 의해 기본창문(왼쪽)이 변경된것을 보여준다.

그림 4-5. Modify대화칸의 입력값으로 크기변경되는 기본창문

Mainline

```
1 /* kdbdata.cpp */
2 #include <kapplication.h>
3 #include <kcmdlineargs.h>
4 #include "mainwidget.h"
5
6 int main(int argc, char **argv)
7 {
8      KCmdLineArgs::init(argc, argv,  "kdbdata",
9               "KDialogBase demo",  "0.0");
10     KApplication app;
11     MainWidget mainwidget;
12     mainwidget.show();
13     app.setMainWidget(&mainwidget);
14     return(app.exec());
15 }
```

프로그람의 기본함수는 11행에서 MainWidget를 만들고 13행에서 그것을 응용프로그람의 기본창문으로 설정한다.

KApplication객체는 10행에서 인수없이 만들어진다. 이것은 8행에서 KCmdLineArgs클라스의 정적init()메쏘드가 호출되므로 가능하다. KCmdLineArgs클라스는 지령행인수와 다른 정보를 보관하여 응용프로그람의 다른 부분에서 사용할수 있게 한다.

MainWidget머리부파일

```
1 /* mainwidget.h */
2 #ifndef MAINWIDGET_H
3 #define MAINWIDGET_H
4
5 #include <qwidget.h>
6 #include <qpushbutton.h>
7 #include <qlabel.h>
8
9 class MainWidget: public QWidget
10 {
11     Q_OBJECT
12 public:
13     MainWidget(QWidget *parent=0,const char *name=0);
```

```
14 private:
15     QLabel *label;
16     QPushButton *button;
17 private slots:
18     void popupKdb();
19     void slotSettings(QString &,int,int);
20 };
21
22 #endif
```

이 창문부품은 프로그람의 기본창문이다. 그것은 오직 1개의 단추와 1개의 표식자를 포함한다. popupKdb()처리부는 대화칸을 펼치는데 사용된다. 처리부 slotSettings()는 대화칸이 돌려준 값들을 받아들인다.

MainWidget

```
1 /* mainwidget.cpp */
2 #include "mainwidget.h"
3 #include "modify.h"
4 #include <qpushbutton.h>
5 #include <kdialogbase.h>
6
7 MainWidget::MainWidget(QWidget *parent,const char *name)
8      : QWidget(parent,name)
9 {
10     setMinimumSize(200,140);
11
12     QString str("Modify Me");
13     label = new QLabel(str,this);
14     label->setAlignment(Qt::AlignCenter);
15     label->setGeometry(50,20,100,40);
16
17     button = new QPushButton("Modify",this);
18     button->setGeometry(50,80,100,40);
19     connect(button,SIGNAL(clicked()),
20             this,SLOT(popupKdb()));
21     resize(10,10);
22 }
```

```
23 void MainWidget::popupKdb()
24 {
25    Modify *modify = new Modify(this,  "modify");
26    connect(
27            modify,SIGNAL(signalSettings(QString &,int,int)),
28            this,SLOT(slotSettings(QString &,int,int)));
29    modify->exec();
30    delete modify;
31 }
32 void MainWidget::slotSettings(QString &str,
33            int height,int width)
34 {
35    resize(width,height);
36    label->setText(str);
37 }
```

이 창문부품은 프로그람의 기본창문으로 사용된다. 그것은 오직 표식자와 단추를 포함한다. 단추는 대화칸을 펼치는데 사용된다.

18∼20행은 단추를 만들어 창문에 배치하고 그것의 clicked()신호를 국부처리부 popupKbd()에 련결한다. 23행에서 시작하는 popupKbd()처리부는 Modify대화칸을 만들고 signalSettings()신호를 slotSettings()국부처리부에 련결한다. exec()를 호출하면 대화칸을 현시하고 그것을 닫을 때까지 기다린다.

32행에서 시작하는 처리부이름 slotSettings()는 인수로서 3개 값을 받아들이고 이 값들을 사용하여 기본창문의 크기, 그리고 기본창문의 표식자에 현시해야 할 본문을 지정한다. 35행의 resize()의 2개 인수는 10행에서 지정한 최소값보다 작은 값을 가질수 없고 그 이상의 값만을 가진다.

Modify머리부파일

```
1 /* modify.h */
2 #ifndef MODIFY_H
3 #define MODIFY_H
4
5 #include <kdialogbase.h>
6 #include <qlineedit.h>
7 #include <qpushbutton.h>
8
9 class Modify: public KDialogBase
```

```
10 {
11     Q_OBJECT
12 public:
13     Modify(QWidget *parent=0,const char *name=0);
14 private:
15     QLineEdit *lineedit;
16     QLineEdit *width;
17     QLineEdit *height;
18 private slots:
19     void slotSendValues();
20 signals:
21     void signalSettings(QString &,int,int);
22 };
23
24 #endif
```

이것은 대화칸의 머리부파일로서 KDialogBase클라스를 직접 계승하며 그 자체의 처리부와 신호를 정의한다.

대화칸의 단추들로부터 응답을 받아들이기 위하여 slotSendValues()라는 처리부를 선언한다. slotSendValues()를 실행할 때마다 새로운 본문과 값들을 가진 signalSettings()신호를 보낸다.

대화칸 그 자체는 KDialogBase를 직접 계승하므로 대부분의 기능은 이미 갖추고있다. 그러므로 재촉문, 자료입력창문부품 그리고 사용자가 새로운 값모임을 지정할 때마다 보내야 할 신호를 추가하려는 경우에는 아래와 같이 코드를 작성한다.

Modify

```
1 /* modify.cpp */
2 #include "modify.h"
3 #include <qlayout.h>
4 #include <qlabel.h>
5
6 Modify::Modify(QWidget *parent,const char *name)
7       : KDialogBase(parent,name,TRUE,  "Modify")
8 {
9       QWidget *mainWidget = new QWidget(this,  "modifymain");
10
11      QVBoxLayout *vLayout = new QVBoxLayout(mainWidget,10);
```

```
12
13     lineedit = new QLineEdit(mainWidget);
14     vLayout->addWidget(lineedit);
15
16     QHBoxLayout *hLayout = new QHBoxLayout();
17     vLayout->addLayout(hLayout);
18
19     QLabel *wLabel = new QLabel("width:",this);
20     wLabel->setAlignment(Qt::AlignCenter);
21     hLayout->addWidget(wLabel);
22     width = new QLineEdit(mainWidget);
23     width->setMaximumWidth(50);
24     hLayout->addWidget(width);
25     QLabel *hLabel = new QLabel("height:  ",this);
26     hLabel->setAlignment(Qt::AlignCenter);
27     hLayout->addWidget(hLabel);
28     height = new QLineEdit(mainWidget);
29     height->setMaximumWidth(50);
30     hLayout->addWidget(height);
31
32     connect(this,SIGNAL(okClicked(void)),
33             this,SLOT(slotSendValues(void)));
34     connect(this,SIGNAL(applyClicked(void)),
35             this,SLOT(slotSendValues(void)));
36
37     setMainWidget(mainWidget);
38 }
39 void Modify::slotSendValues()
40 {
41     QString text = lineedit->text();
42     int w = (width->text()).toInt();
43     int h = (height->text()).toInt();
44     emit signalSettings(text,h,w);
45 }
```

6행에서 시작하는 구성자는 기초클라스에로 인수를 넘긴다. 이행금지대화칸을 지정하고

창문에 제목을 할당하는 두개 인수가 기초클라스호출에 추가된다.

9행에서 빈 창문부품이 만들어진다. 이 창문부품은 11행에서 만들어진 수직칸배치에 의해 배치된다. QLineEdit창문부품은 13행과 14행에서 수직칸의 꼭대기에 삽입된다. 수평칸은 16행에서 창조되고 수직창문부품의 두번째 칸에 들어가는 창문부품들을 배치하는데 사용된다. 이 수평칸은 20～30행에서 QLabel과 QLineEdit창문부품들로 채워진다. 37행에서 창문부품은 대화칸에 기본창문으로서 추가된다.

기정단추들을 선택하므로 대화칸에 나타나는 단추들은 대화칸에 포함된 OK, Cancel 및 Apply단추들이다. Cancel단추가 선택될 때마다 대화칸은 닫긴다. 이 실례에서 Cancel단추에 응답하여 얻어지는 작용은 없고 그것의 신호는 무시된다. 32와 34행에서 2개의 connect()메쏘드호출은 slotSendValues()라는 처리부를 실행하게 한다. 련결들은 처리부와 신호들이 모두 같은 객체에 있고 신호들이 계승되고 처리부가 국부적으로 정의되므로 this로부터 오고 this에로 돌아간다.

39행에서 시작하는 처리부메쏘드는 사용자가 입력한 정보를 수집하고 자료를 사용하여 신호를 발생시킨다. 행편집칸의 본문은 표식자의 제목을 변경하는데 사용되므로 그것은 같은 형식으로 될수 있다. QLineEdit객체들이 돌려주는 너비와 높이는 QString객체들이지만 toInt()호출에 의해 int값들로 변환된다.

제7절. KDialogBase자료호출방법

다음 실례프로그람은 앞의 실례를 변경한것이다. 가끔 프로그람은 처리부와 신호를 리용하여 값들을 주고받을 대신에 대화칸을 리용하여 얻는것이 편리하다. 이 수법은 대화칸이 닫길 때 대화칸으로부터 자료를 얻을 필요가 있는 경우에 적용한다.

앞의 실례에서 Apply단추는 창문을 닫지 않고 응용프로그람에 자료를 제공한다. 이 실례는 그림 4-6에 보여준 창문에서 Apply단추를 삭제하고 모든 처리부와 신호들을 제거한다.(다만 대화칸을 펼치는 기본함수는 제외한다.)
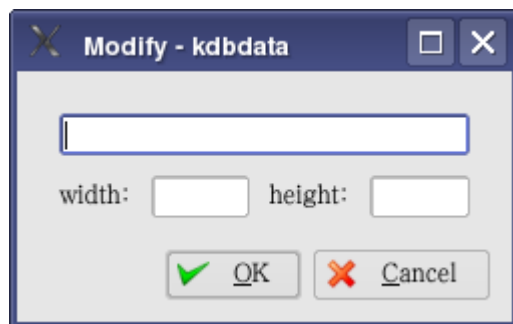
그림 4-6. 처리부들을 사용하지 않고 자료입력

그러기 위하여 다음과 같이 modify.cpp의 6행과 7행을 변경한다.

```
Modify::Modify(QWidget *parent,const char*name)
        : KDialogBase(parent,name,TRUE,  "Modify",Ok | Cancel)
```

다음 단계는 mainwidget.cpp의 23~37행을 삭제하고 그것들을 다음 메쏘드로 교체한다.

```
void MainWidget::popupKdb()
{
   Modify *modify = new Modify(this, "modify");
   modify->exec();
   if(modify->result() == TRUE) {
      QString text = modify->getText();
      int height = modify->getHeight();
      int width = modify->getWidth();
      resize(width,height);
      label->setText(text);
   }
   delete modify;
}
```

대화칸이 닫길 때 exec()메쏘드는 완료한다. OK단추가 대화칸을 닫는데 사용되면 result()로부터의 돌림값은 TRUE이고 그렇지 않으면 FALSE이다. 결과가 TRUE이면 사용자입력자료를 창문부품들로부터 끌어내여 resize()와 setText()를 호출하여 현시를 변경하는데 쓰인다. exec()메쏘드는 대화칸을 닫아야 완료하고 Apply단추가 주어지지 않았으므로 창문을 닫지 않는 단추선택을 결정하는데 처리부외에 방법이 없다.

값을 얻기 위하여 다음의 코드를 Modify클라스에 추가한다.

```
QString Modify::getText()
{
   return(lineedit->text());
}
int Modify::getWidth()
{
   return((width->text()).toInt());
}
int Modify::getHeight()
{
   return((height->text()).toInt());
}
```

알아두기: 대화칸으로부터 정보를 얻는 방법은 많다. 실례로 대화칸에 struct의 주소를 넘기고 거기에 자료를 채울수 있다. 혹은 처리부,신호결합을 사용하여 직접 값들을 읽을수 있다. 심지어 대화칸을 가지고 후에 자기 프로그람에서 사용할수 있는

환경구성파일에 그 출력을 쓸수 있다.

제8절. KDialogBase의 파생클라스 KMessageBox

대화칸의 매우 일반적인 형태는 한행이나 두행의 본문을 현시하고 사용자가 간단히 《예》라고 응답하거나 전혀 응답하지 않거나 간단히 대화칸을 완료하는 단추를 누르는것이다. KDialogBase클라스는 통보대화칸을 작성하는데 특별히 적합한 구성자를 가지고 있고 KMessageBox클라스는 이 특수구성자를 리용하여 일반적으로 사용하는 통보대화칸그룹을 실현한다.

이 통보칸들은 모두 이행금지대화칸으로서 이동하기전에 사용자의 응답을 요구한다. 또한 매개 통보칸은 간단한 함수호출에 의해 펼쳐지고 사용자가 응답하여 통보칸을 닫은 다음에야 호출측에로 되돌아간다. 이것은 간단히 자기 코드의 어떤 점에서 정적함수호출을 삽입하는 방법으로 프로그람을 작성할수 있다.

다음 실례는 9개의 통보칸을 보여준다. 그림 4-7에 보여준 기본창문은 9개 대화칸에 대응하는 단추를 하나씩 가지고있다. 대화칸으로부터 응답이 있을 때마다 창문의 밑에 본문이 갱신되는 표식자가 있다. 그림은 마지막선택이 Yes단추라는것을 보여준다.
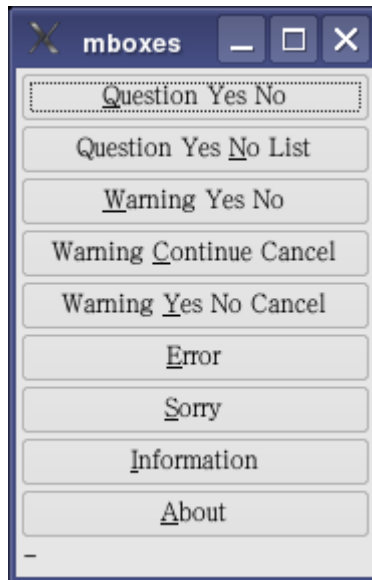
그림 4-7. 단추를 선택하여 통보칸을 현시

Mainline

```
1 /* main.cpp */
2 #include <kapplication.h>
3 #include <kcmdlineargs.h>
4 #include "mboxes.h"
5
```

```
6 int main(int argc, char **argv)
7 {
8     KCmdLineArgs::init(argc, argv,  "mboxes",
9           "Message Boxes",  "0.0");
10    KApplication app;
11    Mboxes mboxes;
12    mboxes.show();
13    app.setMainWidget(&mboxes);
14    return(app.exec());
15 }
```

기본함수는 Mboxes객체를 만들어 응용프로그람의 기본창문으로 설치한다.

Mboxes머리부파일

```
1 /* mboxes.h */
2 #ifndef MBOXES_H
3 #define MBOXES_H
4
5 #include <qwidget.h>
6 #include <qlabel.h>
7
8 class Mboxes: public QWidget
9 {
10    Q_OBJECT
11 public:
12    Mboxes(QWidget *parent=0,const char *name=0);
13 private:
14    QLabel *label;
15 private slots:
16    void button1();
17    void button2();
18    void button3();
19    void button4();
20    void button5();
21    void button6();
22    void button7();
23    void button8();
```

```
24    void button9();
25 };
26
27 #endif
```

클라스선언에는 9개의 단추 매개에 대하여 처리부를 하나씩 포함하고 창문의 아래에 놓이는 표식자를 포함한다.

Mboxes

```
1 /* mboxes.cpp */
2 #include "mboxes.h"
3 #include <qpushbutton.h>
4 #include <kmessagebox.h>
5 #include <qlayout.h>
6
7 Mboxes::Mboxes(QWidget *parent,const char *name)
8       : QWidget(parent,name)
9 {
10    QPushButton *button;
11    QVBoxLayout *layout = new QVBoxLayout(this,3);
12
13    button = new QPushButton("Question Yes No",this);
14    layout->addWidget(button);
15    connect(button,SIGNAL(clicked()),this,SLOT(button1()));
16
17    button = new QPushButton("Question Yes No List",this);
18    layout->addWidget(button);
19    connect(button,SIGNAL(clicked()),this,SLOT(button2()));
20
21    button = new QPushButton("Warning Yes No",this);
22    layout->addWidget(button);
23    connect(button,SIGNAL(clicked()),this,SLOT(button3()));
24
25    button =
26            new QPushButton("Warning Continue Cancel",this);
27    layout->addWidget(button);
28    connect(button,SIGNAL(clicked()),this,SLOT(button4()));
```

```
29
30     button = new QPushButton("Warning Yes No Cancel",this);
31     layout->addWidget(button);
32     connect(button,SIGNAL(clicked()),this,SLOT(button5()));
33
34     button = new QPushButton("Error",this);
35     layout->addWidget(button);
36     connect(button,SIGNAL(clicked()),this,SLOT(button6()));
37
38     button = new QPushButton("Sorry",this);
39     layout->addWidget(button);
40     connect(button,SIGNAL(clicked()),this,SLOT(button7()));
41
42     button = new QPushButton("Information",this);
43     layout->addWidget(button);
44     connect(button,SIGNAL(clicked()),this,SLOT(button8()));
45
46     button = new QPushButton("About",this);
47     layout->addWidget(button);
48     connect(button,SIGNAL(clicked()),this,SLOT(button9()));
49
50     label = new QLabel("-",this);
51     layout->addWidget(label);
52     resize(10,10);
53 }
54 void Mboxes::button1()
55 {
56     int result = KMessageBox::questionYesNo(this,
57             "Are you sure you want to delete\nall "
58             "the files in this directory?  ",
59             "questionYesNo");
60     switch(result) {
61             case KMessageBox::Yes:
62                     label->setText(QString("Yes"));
63                     break;
```

```
64          case KMessageBox::No:
65                  label->setText(QString("No"));
66                  break;
67      }
68 }
69 void Mboxes::button2()
70 {
71      QStringList list;
72      list.append("fork");
73      list.append("spoon");
74      list.append("knife");
75      int result = KMessageBox::questionYesNoList(this,
76          "Are you sure you want to delete\nall "
77          "the items shown in the list?  ",
78          list,
79          "questionYesNoList");
80      switch(result) {
81          case KMessageBox::Yes:
82                  label->setText(QString("Yes"));
83                  break;
84          case KMessageBox::No:
85                  label->setText(QString("No"));
86                  break;
87      }
88 }
89 void Mboxes::button3()
90 {
91      int result = KMessageBox::warningYesNo(this,
92          "Reset all status codes?  ",
93          "warningYesNo");
94      switch(result) {
95          case KMessageBox::Yes:
96                  label->setText(QString("Yes"));
97                  break;
98          case KMessageBox::No:
```

```
99                  label->setText(QString("No"));
100                 break;
101   }
102 }
103 void Mboxes::button4()
104 {
105   int result = KMessageBox::warningContinueCancel(this,
106         "Overwrite the existing file?  ",
107         "warningContinueCancel",
108   QString("Overwrite"));
109   switch(result) {
110         case KMessageBox::Continue:
111                 label->setText(QString("Continue"));
112                 break;
113         case KMessageBox::Cancel:
114                 label->setText(QString("Cancel"));
115                 break;
116   }
117 }
118 void Mboxes::button5()
119 {
120   int result = KMessageBox::warningYesNoCancel(this,
121         "Quitting without saving the file could result\n"
122         "in loss of data. Save before quitting?",
123         "warningYesNoCancel");
124   switch(result) {
125         case KMessageBox::Yes:
126                 label->setText(QString("Yes"));
127                 break;
128         case KMessageBox::No:
129                 label->setText(QString("No"));
130                 break;
131         case KMessageBox::Cancel:
132                 label->setText(QString("Cancel"));
133                 break;
```

```
134   }
135 }
136 void Mboxes::button6()
137 {
138   KMessageBox::error(this,
139             "Unable to save configuration data.");
140 }
141 void Mboxes::button7()
142 {
143   KMessageBox::sorry(this,
144             "The file you specified contains no data.");
145 }
146 void Mboxes::button8()
147 {
148   KMessageBox::information(this,
149             "Pressing Esc will clear the window.  ");
150 }
151 void Mboxes::button9()
152 {
153   KMessageBox::about(this,
154             "This is a simple about-box that can\n"
155             "contain several lines of text");
156 }
```

기본창문으로 쓰이는 창문부품은 9개의 단추와 1개의 표식자를 만들고 그것들을 수직칸에 묶어놓는다. 매개 단추는 하나의 통보칸을 만들어 현시하는 국부처리부에 련결된 clicked()신호를 가진다.

수직칸은 11행에서 만들어진다. 13～51행은 기본창문을 이루는 9개 단추와 표식자를 만든다.

13～15행은 단추를 만들고 54행에서 그것을 처리부 button1()에 련결한다. 56행에서 메쏘드 questionYesNo()호출에 의해 그림 4-8에 보여주는 통보칸을 현시하고 사용자가 응답하기를 기다린다. 59행의 문자렬파라메터는 통보칸창문의 제목이다. 함수로부터 돌아온 값은 선택된 단추를 결정한다. 다른 모든 통보칸들로부터의 모든 돌림값들은 KMessageBox클라스에서 정의된다.
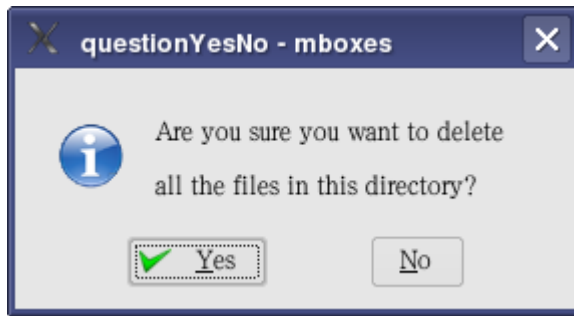
그림 4-8. Yes/No질문통보칸

알아두기: 통보칸들로부터의 돌림값은 론리형이 아니다. 그것들은 TRUE 또는 FALSE를 검사하는 식에 사용될수 없다.

61행의 switch문은 질문 YesNo()로부터의 돌림값이 Yes인가 No인가를 결정한다. 기본창문바닥의 표식자는 결과를 표시하는 문자렬로 갱신된다.

모든 통보칸들은 본문행의 길이와 현시행수를 자체로 결정하므로 그 크기와 모양은 자체로 관리한다. questionYesNo()통보칸의 본문은 57행과 58행에서 정의된다. 본문은 문자렬을 분리할 때마다 삽입되는 '\n'문자를 가지는 단일문자렬로 정의된다. C++는 개별적인 행에 있는 문자렬들을 하나의 큰 문자렬로 결합하여 코드에서 본문을 간단히 쓸수 있다. 17～19행은 단추를 만들고 69행에서 시작하는 처리부 button2()에 그것을 련결한다. 이 통보칸은 75행에서 만들어지고 그림 4-9와 같이 현시되며 앞의 실례와 같이 예 또는 아니 질문을 제기하고 또한 항목목록을 현시하는 창문을 포함한다. 항목그룹을 가진 질문을 제시할 필요가 있을 때 이 통보칸을 사용한다. 이때 사용자가 항목을 추가하거나 삭제하는 방법은 없으며 오직 목록전체의 허가나 거절로 된다.
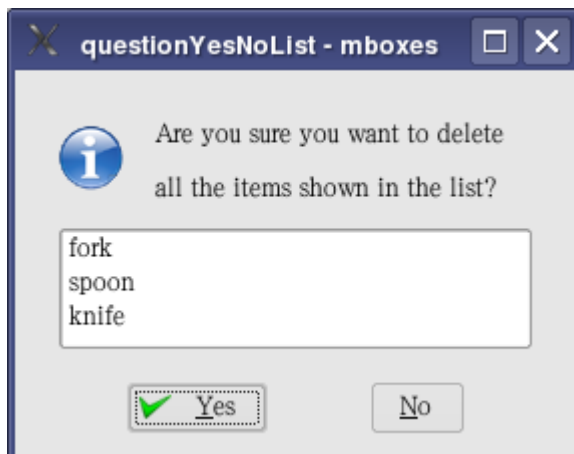
그림 4-9. Yes/No목록질문통보칸

현시목록은 71행에서 만들어지고 72～74행에서 3개 문자렬들로 채워지는 QStringList객체이다. 이 문자렬목록은 78행에서 인수로서 사용되고 79행의 문자렬은 창문제목을 지정한다.

21～23행에서는 단추를 만들고 89행에서 시작하는 처리부 button3()에 그것을 련결한다. 91행에서 warningYesNo()호출은 56행에서 questionYesNo()호출과 거의 같다. 그라프와 본문이 현시되였다는것만 다르다. 경고통보칸을 그림 4-10에 보여준다.

그림 4-10. Yes/No경고통보칸

25～28행은 단추를 만들고 103행에서 시작하는 처리부 button4()에 그것을 련결한다. warningContinueCancel()통보칸은 사용자에게 어떤 조작을 시작하려고 한다는것을 알려주고 사용자가 계속하겠는가 중지하겠는가를 결정할수 있도록 설계한다. 그림 4-11에 보여주는것처럼 이 실례는 현존파일을 덧쓰기한다는것을 사용자에게 경고한다.
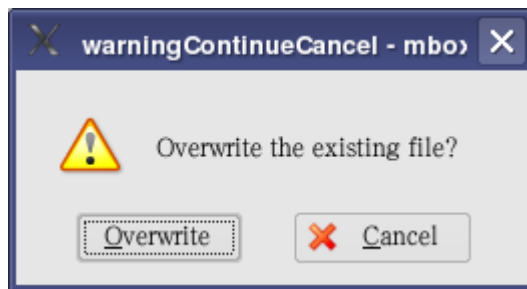
그림 4-11. Continue/Cancel경고통보칸

일부 통보칸들은 사용자가 단추표식자들을 무시하게 한다. 이 실례의 108행에서 인수로서 사용한 QString은 기정의 Continue로부터 Overwrite로 단추표식자를 변경한다. 즉 단추제목의 변경으로 돌림값은 달라지지 않고 110행의 case문은 Continue값과 대조되여 제목이 Overwrite로 변경된다.

간단히 예 또는 아니라는 대답이 적합하지 않을 때가 있다. 30～32행은 단추를 만들고 118행에서 시작하는 처리부 button5()에 그것을 결합한다. 이런 상황은 자주 발생한다. 실례로 통보문 "Preparing to delete files. Do you wish to delete subdirectories also ?"을 고찰하자. 사용자는 파일들과 함께 삭제하려는 등록부들을 지정하여야 한다. 120행에서 warningYesNoCancel()호출에 의해 만들어진 통보칸을 그림 4-12에서 보여준다.
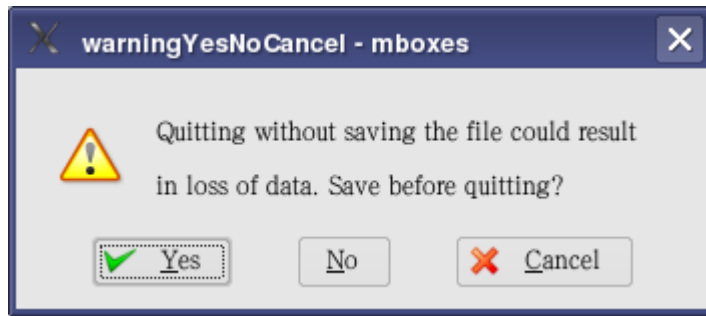
그림 4-12. Yes/No/Cancel경고통보칸

34～35행은 단추를 만들고 136행에서 시작하는 처리부 button6()에 그것을 결합한다. 138행에서 error()호출은 그림 4-13에 보여주는 통보칸을 만든다. 이 통보칸은 오직 자기 프로그람이 정상적인 일을 할수 없을 때에만 사용하려고 하므로 돌림값을 가지지 않는다.
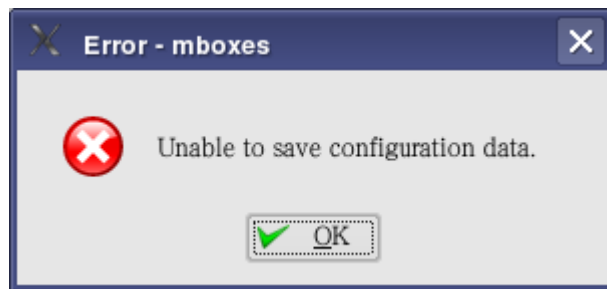
그림 4-13. Error통보칸

38-40행은 단추를 만들고 141행에서 시작하는 처리부 button7()에 그것을 련결한다. 함수 sorry()는 143행에서 호출되여 그림 4-14에 보여주는 창문을 현시한다. 이것은 프로그람이 실행을 계속할수 없는 경우를 위한것으로서 프로그람에서 조종할수 없다.
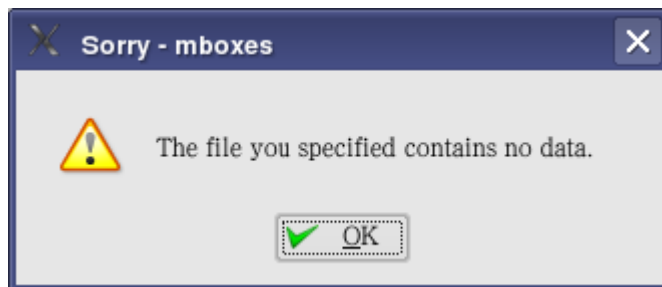
그림 4-14. Sorry통보칸

42～44행은 단추를 만들고 146행에서 시작하는 처리부 button8()에 그것을 련결한다. 함수 information()은 148행에서 호출되고 그림 4-15에 보여주는 창문을 현시한다. 이것은 처리에 영향을 주지 않는 정보를 포함하기 위한것이므로 사용자가 알아야 하는 경우가 있다.
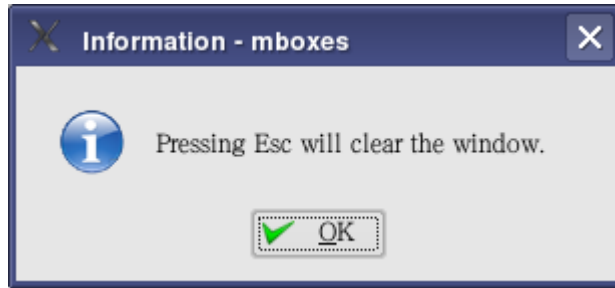
그림 4-15. Information통보칸

46～48행은 단추를 만들고 151행에서 처리부 button9()에 그것을 련결한다. 이것은 가장 간단한 통보칸으로서 도형장식을 하지 않고 값을 돌려주지 않는다. 그림 4-16는 153행에서 about()호출에 의해 만들어진 통보칸을 보여준다. about()로 시작하는 함수는 매우 간단한 About칸으로 사용될수 있다.
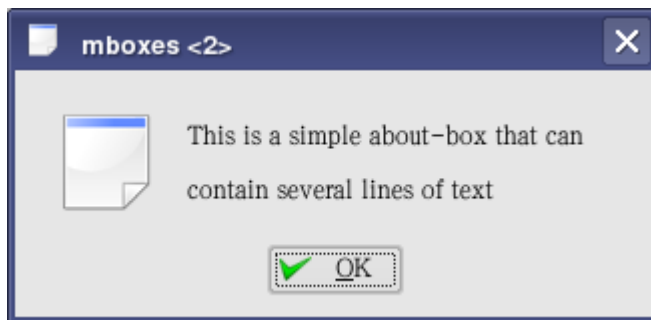
그림 4-16. 간단한 통보칸

요 약

대화칸을 만드는 방법은 많다. 대화칸의 기본창문은 언제나 임의의 한개 창문부품이다. 그러므로 대화칸을 현시하고 사용자입력을 검색하며 창문을 배치하는 기구를 따로따로 써넣을수 있다. 이 장에서 설명한 주요 부분은 다음과 같다.

· 대화칸에 관한 특별한 요구조건이 있다면 기초클라스로서 QDialog와 KDialog를 사용하여 대화칸을 창조할수 있다.

· KDialogBase는 창문부품을 삽입하는데 사용할수 있는 표준단추들과 배치관리기들을 제공함으로써 대화칸의 구축을 돕는다.

· 대화칸은 사용자입력자료를 포함한 신호를 보낼수 있고 프로그람은 처리부를 사용하여 자료를 받을수 있다.

· KMessageBox클라스는 정적함수집합으로서 매개 정적함수는 KDialogBase를 사용하여 표준대화칸을 구축한다.

대부분의 프로그람은 대화칸을 사용한다. 대화칸은 단추를 리용하여 펼칠수 있고(이 장에서 설명) 프로그람내의 사건으로부터 펼칠수도 있으며(파일에 오유쓰기 등) 또는 사용자가 차림표나 도구띠를 선택할 때 펼쳐질수 있다. 프로그람의 형식을 표준화하는 한가지 방법은

이미 짜넣져있는 내부대화칸을 사용하는것이다. 이 장은 간단한 내부대화칸의 일부를 설명하였으며 다음 장은 더 복잡한 대화칸들을 설명한다.

제5장. 미리 정의된 대화칸들

학습내용

표준About대화칸의 만들기,

사용자가 파일이름을 선택할수 있게 하기,

한개 대화칸안에서 여러개 창문부품들의 페지만들기,

작업의 완성정도를 현시하기.

대단히 많은 대화칸이 KDE와 Qt에서 정의되여있다. 이 장은 몇가지 일반대화칸을 보여준다. 대화칸클라스를 기초클라스로 하여 사용자대화칸을 만들고 대화칸의 실례를 만들고 필요한 선택을 하여야 한다. 이 장에서는 대화칸의 모든것을 설명하지 않는다. 일부 특수대화칸들은 다른 장들에서 설명한다. 실례로 KFontDialog는 10장《서체》에서, KColorDialog는 11장《색》에서 설명한다.

제1절. About대화칸

완성된 모든 응용프로그람은 About칸을 가진다. About칸은 쏘프트웨어의 계승에 대한 부속정보를 현시하는 창문이다. 이것은 프로그람작성자들이 자기 이름을 표시할수 있는곳이다. About칸은 단추로부터 펼칠수 있으나 차림표선택으로 더 자주 사용한다.

KAboutDialog클라스는 매우 유연하다. 그것은 작성자가 포함하거나 제외할수 있는 선택적인 부분품들의 집합이다. 또한 서로다른 기본외형을 주는 2개의 독립적인 구성자를 가지고있다. 기정으로 나타나는 유일한 단추는 대화칸을 닫는데 쓰이는 OK단추이다. 유연성을 위하여 About칸은 많은 선택을 제공하는 KDialogBase클라스들을 계승한다.(KDialogBase클라스는 4장에서 서술된다.)

KAboutDialog클라스는 2개의 구성자를 가지고있다. 구성자들은 각이한 인수들을 가지는 것과 함께 각이한 방식으로 대화칸을 배치하며 매개 방식은 자체의 항목들을 가지고있다. 구성자I는 표준Qt형식으로서 부모창문부품, 내부이름 그리고 창문이 이행금지인가 아닌가하는 3개의 인수를 가진다.

```
KAboutDialog(QWidget *parent,const char *name=0, bool modal=true)
```

구성자Ⅱ는 여러가지 배치형식들중 하나의 형식으로 대화칸을 현시하는 환경구성에 쓰이는 기발들을 비롯하여 많은 파라메터들(대부분이 기정값을 가진다)을 가지고있다

```
KAboutDialog(int dialogLayout,const QString &caption,
   int buttonMask,int defaultButton,QWidget *parent=0,
   const char *name=0,bool modal=false,bool separator=false,
   const QString &user1=QString::null,
   const QString &user2=QString::null,
   const QString &user3=QString::null)
```

다음 실례는 구성자 I과 II를 모두 리용하여 가능한 배치를 만들고 현시한다. 여기서 보여주는것보다 훨씬 더 많은 결합이 가능하다. 프로그람의 기본창문은 그림 5-1에 보여주며 4개 판의 About대화칸을 펼치는데 쓰이는 단추들로 구성되여있다.

그림 5-1. 4개의 About대화칸을 펼치는데 쓰이는 단추들

ShowAbout머리부파일

```
1 /* showabout.h */
2 #ifndef SHOWABOUT_H
3 #define SHOWABOUT_H
4
5 #include <qwidget.h>
6
7 class ShowAbout: public QWidget
8 {
9     Q_OBJECT
10 public:
11    ShowAbout(QWidget *parent=0,const char *name=0);
12 private slots:
13    void emptyAbout();
14    void simpleAbout();
15    void kdeStandardAbout();
16    void appStandardAbout();
17 };
18
19 #endif
```

이 클라스의 유일한 목적은 필요한 대화칸을 펼치는것이므로 유일한 구성자와 4개 단추에 의해 사용되는 처리부가 있다. 매개 처리부는 서로 다른 판의 About대화칸을 펼친다.

ShowAbout

```
1 /* showabout.cpp */
2 #include <kapplication.h>
3 #include <qpushbutton.h>
4 #include <qlayout.h>
5 #include <kaboutdialog.h>
6 #include <kcmdlineargs.h>
7 #include "showabout.h"
8
9 int main(int argc, char **argv)
10 {
11     KCmdLineArgs::init(argc, argv,  "showabout",
12             "About Boxes","0.0");
13     KApplication app;
14     ShowAbout showabout;
15     showabout.show();
16     app.setMainWidget(&showabout);
17     return(app.exec());
18 }
19
20 ShowAbout::ShowAbout(QWidget *parent,const char *name)
21     : QWidget(parent,name)
22 {
23     QPushButton *button;
24     QVBoxLayout *box = new QVBoxLayout(this);
25
26     button = new QPushButton("Empty",this);
27     box->addWidget(button);
28     connect(button,SIGNAL(clicked()),
29             this,SLOT(emptyAbout()));
30
31     button = new QPushButton("Simple",this);
32     box->addWidget(button);
33     connect(button,SIGNAL(clicked()),
34             this,SLOT(simpleAbout()));
35
```

```
36     button = new QPushButton("KDE Standard",this);
37     box->addWidget(button);
38     connect(button,SIGNAL(clicked()),
39             this,SLOT(kdeStandardAbout()));
40
41     button = new QPushButton("App Standard",this);
42     box->addWidget(button);
43     connect(button,SIGNAL(clicked()),
44             this,SLOT(appStandardAbout()));
45
46     resize(10,10);
47     box->activate();
48 }
49 void ShowAbout::emptyAbout()
50 {
51     KAboutDialog *about = new KAboutDialog(0,  "about");
52     about->exec();
53 }
54 void ShowAbout::simpleAbout()
55 {
56     KAboutDialog *about = new KAboutDialog(0,  "about");
57
58     about->setCaption("Simple About Configuration");
59     about->setVersion("Version 0.0.1");
60
61     QPixmap logo;
62     if(logo.load("tinylogo.png"))
63             about->setLogo(logo);
64
65     about->setAuthor("Bertha D Blues",
66             "bertha@belugalake.com",
67             "http://www.belugalake.com",
68             "Mallet Operator");
69
70     about->setMaintainer("Tony Stryovie",
```

```
71            "stryovie@belugalake.com",
72            "http://www.belugalake.com",
73            "Finder of Lost Code");
74
75    about->addContributor("Walter Heater",
76            "heat@belugalake.com",
77            "http://www.belugalake.com",
78            "Asker of Questions");
79
80    about->exec();
81 }
82 void ShowAbout::kdeStandardAbout()
83 {
84    KAboutDialog *about = new KAboutDialog(
85            KAboutDialog::AbtKDEStandard,
86            "KDE Standard Configuration",
87            KDialogBase::Ok | KDialogBase::Help,
88            KDialogBase::Ok,
89            this,
90            "about",
91            TRUE);
92
93    about->setTitle("The example that is all about About");
94    about->setCaption("KDE Standard About");
95    about->setImage("penguin1.png");
96    about->setImageBackgroundColor(QColor("red"));
97    about->setImageFrame(TRUE);
98
99    about->addTextPage("Purpose",
100           "This program is intended to provide an "
101           "example that\ndemonstrates how to use "
102           "the KAboutDialog.");
103   about->addTextPage("Version",
104           "Version 0.0.1 pre-alpha experimental.\n"
105           "Saturday, April 1, 2000");
```

```
106
107   about->exec();
108 }
109 void ShowAbout::appStandardAbout()
110 {
111   KAboutDialog *about = new KAboutDialog(
112           KAboutDialog::AbtAppStandard,
113           "App Configuration",
114           KDialogBase::Ok,
115           KDialogBase::Ok,
116           this,
117           "about",
118           TRUE);
119
120   about->setTitle("The example that is about About");
121   about->setProduct("ShowAbout",
122           "0.0.1 Pre-Alpha",
123           "Bertha D Blues",
124           "Saturday, April 1, 2000");
125
126   about->addTextPage("Purpose",
127           "This program is intended to provide an "
128           "example that\ndemonstrates how to use "
129           "the KAboutDialog.");
130   about->addTextPage("Version",
131           "Version 0.0.1 pre-alpha experimental.\n"
132           "Saturday, April 1, 2000");
133
134   about->exec();
135 }
```

9～18행은 프로그람의 기본함수로서 ShowAbout클라스의 실례를 만들어 기본창문에서 사용할 창문부품으로 삽입하고 완료를 기다린다.

16행에서 시작하는 ShowAbout구성자는 수직칸에 누름단추들을 배치한다. 4개 단추는 각각 KAboutDialog의 한개 판을 현시하는 처리부들중 하나와 련결된다.

49행의 처리부 emptyAbout()는 그림 5-2에 보여주는 빈 KAboutDialog를 현시한다. 이 기

정판에 나타나는 유일한것은 대화칸을 닫는 OK단추와 일반적으로 응용프로그람용 줄임화상(logo graphics)을 보관하는 위치이다. 51행에서 창조된 대화칸은 이행금지대화칸이므로 52행에서 호출하는 메쏘드 exec()는 사용자가 창문을 닫을 때까지 되돌려지지 않는다.

그림 5-2. 빈 KAboutDialog

54행에서 시작하는 처리부 simpleAbout()는 그림 5-3에 보여준 빈 KAboutDialog를 만드는데 사용한것과 같은 구성자를 리용한다. KAboutDialog메쏘드는 현시가능한 정보를 삽입하기 위하여 호출한다. 표 5-1은 구성자I대화칸에 의해 사용하도록 특별히 설계된 메쏘드들을 보여준다. 58행에서 setCaption()호출은 창문제목띠의 제목을 정의하며 setVersion()호출은 창문꼭대기에 삽입할 판번호를 지정한다.

그림 5-3. 개발자들의 이름을 구성하는 KAboutDialog

표 5-1. 구성자I 용 KAboutDialog메쏘드

메쏘드	목적
setLogo()	줄임화상으로 현시하려는 픽스매프화상을 지정한다.
setAuthor()	이름, 전자우편주소, URL, 쏘프트웨어저자에 의해 처리되는 일감을 지정한다.
setContributor()	개별적인 기부자의 이름, 전자우편주소, URL 그리고 쏘프트웨어에 수행한 일감을 지정한다.
setMaintainer()	개별적인 관리자의 이름, 전자우편주소, URL , 그리고 쏘프트웨어를 지원하기 위하여 수행한 일감을 지정한다.
setVersion()	현재 쏘프트웨어판을 지정한다.

61행에서 창조한 QPixmap는 62행의 tinylogo.png라는 파일로부터 화상정보를 적재한다. 63행의 setLogo()호출은 화상을 창문의 왼쪽웃구석에 현시할 줄임화상으로 설정한다.

65～78행은 setAuthor(), setMaintainer() 및 addContributor()호출에 의해 이름과 주소를 추가한다. 한명의 저자와 1명의 관리자만 있을수 있으나 공헌자들이 많을수 있다. 매개 이름을 삽입하는 각이한 메쏘드들이 있으며 모든 메쏘드들은 같은 파라메터모임을 가지고있다. 첫 문자렬은 이름, 둘째는 개인의 URL, 3번째는 전자우편주소 그리고 마지막이 그 사람이 개발팀에 수행한 일의 설명이다. 일단 현시되면 URL과 전자우편주소 모두는 능동으로 된다. 즉 그것들을 찰칵하여 전자우편을 보내거나 웹페지를 실을수 있다.

82행에서 시작하는 처리부 kdeStandardAbout()은 구성자Ⅱ를 사용하여 그림 5-4에 보여주는 사용자정의대화칸을 만든다. 85행에서 AbtKDEStandard기발모임은 창문현시를 구성하는 요소들을 지정한다. 표 5-2에는 포함하는 기발들과 요소들을 보여준다. 87행의 인수들은 기초클라스 KDialogBase에 어느 단추들을 포함하는가를 알리는데 사용된다. 이 실례에는 자동적으로 창문을 닫는 OK단추와 Help단추가 있다. 88행에 지정된 단추는 대화칸이 처음으로 열릴 때 선택되는것이다.

참고: 4장의 KDialogBase에서는 OK와 Help단추의 의미와 창조방법을 설명한다.
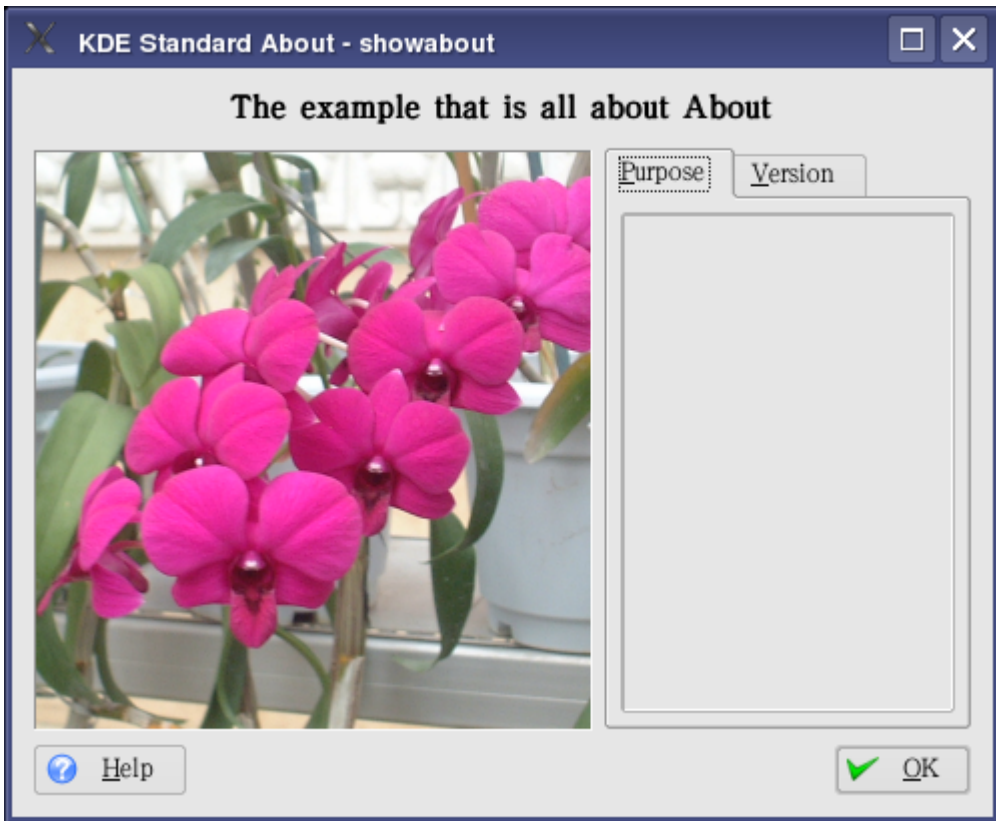
그림 5-4. 하나의 화상과 본문페지를 가지는 KAboutDialog

표 5-2. 창문현시를 구성하는 KAboutDialog기발

기발이름	설명
AbtAppStandard	AbtTabbed, AbtTitle 그리고 AbtProduct의 결합을 허용한다.
AbtImageAndTitle	AbtPlain과 AbtImageOnly의 결합을 허용한다..
AbtImageLeft	창문의 왼쪽에 화상을 현시한다.
AbtImageOnly	다른 항목이 없으므로 화상을 중심에 현시한다.
AbtImageRight	창문의 오른쪽에 화상을 현시한다.
AbtKDEStandard	AbtTabbed, AbtTitle 그리고 AbtImageLeft의 결합을 허용한다.
AbtPlain	기정 KAboutDialog구성요소들중 어느것도 현시되지 않는다. 이것은 다른 기발들과 결합하여 대화칸을 사용자정의하는데 사용될 수 있다.
AbtProduce	응용프로그람이름, 판, 저자 그리고 날자를 현시한다.
AbtTabbed	사용자가 절환할수 있는 타브들을 가진 한개 이상 창문들의 집합을 현시한다.
AbtTitle	바로 제목띠아래에 제목을 현시한다.

대화칸에 어느 요소들을 현시해야 하는가를 알리는 기발들이 설정된 다음에도 요소들을 제공하여야 한다. 표 5-3에는 구성자Ⅱ를 사용할 때 가능한 메쏘드들을 보여준다.

표 5-3. 구성자Ⅱ용 KAboutDialog메쏘드들

메쏘드	목적
addContainer()	본문과 화상현시에 사용할수 있는 KAboutContainer를 추가한다.
addContainerPage()	KAboutContainer를 타브페지로서 추가한다. 그것을 리용하여 본문과 화상을 현시할수 있다.
addPage()	빈페지를 타브페지집합에 추가한다. 그것을 리용하여 현시하고 싶은 창문부품을 포함할수 있다.
addTextPage()	타브페지집합에 본문페지를 추가한다.
setImage()	줄임화상으로 현시하려는 화상을 지정한다.
setImageBackground()	화상을 현시할 때 배경으로 채우는데 사용할 색을 지정한다.
setImageFrame()	화상주위의 틀을 허용 또는 금지한다. 기정은 허용된다.
setProduct()	응용프로그람이름, 판, 저자, 날자를 정의하는 4개 문자렬을 설정한다.
setTitle()	창문꼭대기(바로 제목띠아래)에 제목을 삽입한다.

93행의 setTitle()호출은 기발설정이 AbtTitle을 포함하기때문에 또한 현시하려는 제목을 제공하는데 필요하다. 94행의 setCaption()호출은 꼭대기에 있는 제목띠의 제목본문을 설정하기 위하여 기초클라스에 넘기는 호출이다.

95～97행의 메쏘드호출은 줄임화상으로 사용하려는 화상파일의 이름, 화상의 배경색 그리고 화상이 경계선을 가지는가 아닌가를 지정한다. 배경색은 그림 5-4에 이미 보여준것처럼 창문의 빈 구역들과 화상의 투명부분의 채움색이다. 기정적으로 경계선을 그리지만 메쏘드 setImageFram()으로 그것을 허용하지 않을수 있다.

메쏘드 addTextPage()는 여러번 사용할수 있는데 타브창문에 추가하려는 매개 페지에 대하여 한번씩 호출한다. 99행과 103행의 실례들에서 알수 있는것처럼 오직 인수들은 타브용 표식자로 될 문자렬과 본문본체로 되여야 할 본문이다. 여러행에 본문을 형식화하려면 새행기호 '\n'을 삽입해야 한다. 109행에서 시작하는 appStandardAbout()라는 처리부는 그림 5-5에 보여주는 대화칸을 생성한다.
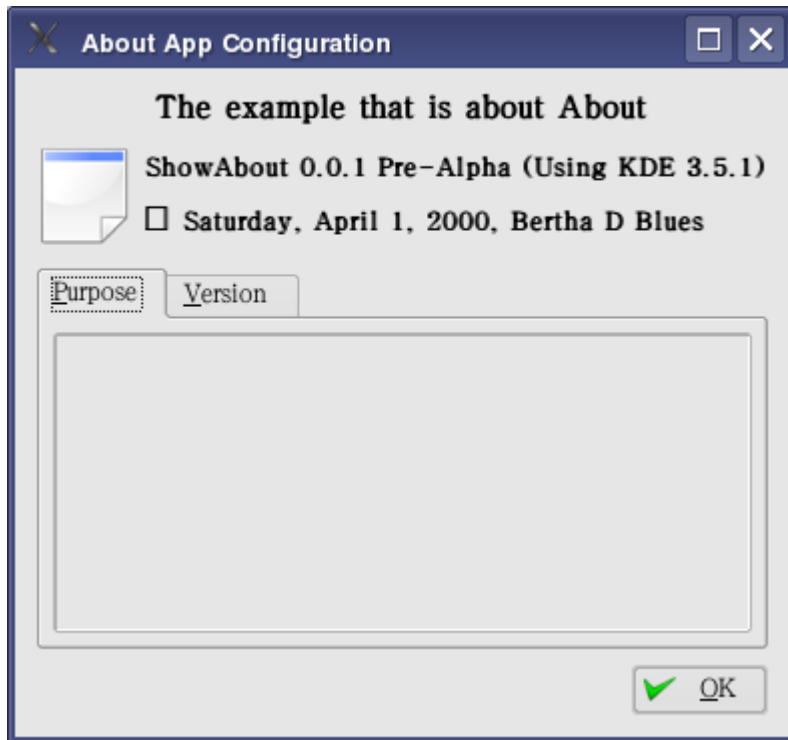
그림 5-5. 머리부와 타브페지들을 가지는 KAboutDialog

창문꾸미기의 본문은 2개의 메쏘드호출에 의해 설정된다. 116행의 setTitle()호출은 웃끝의 중심에 배치되는 제목본문을 설정한다. 121행의 setProduct()호출은 본문의 나머지를 설정한다. 두 타브페지는 126행과 130행에서 addTextPage()호출에 의해 추가된다.

제2절. QFileDialog

QFileDialog는 파일 또는 등록부의 이름을 사용자가 선택할수 있게 한다. 선택을 현존파일들로 제한하거나 사용자가 새로운 파일이름을 입력하도록 지정할수 있다. 또한 려파기를 리용하여 파일이름들을 지정된 기준을 만족시키는것들로 제한할수 있다.

ShowFile머리부파일

```
1 /* showfile.h */
2 #ifndef SHOWFILE_H
3 #define SHOWFILE_H
4
5 #include <qwidget.h>
6 #include <qlabel.h>
7 #include <qstring.h>
8
9 class ShowFile: public QWidget
```

```
10 {
11     Q_OBJECT
12 public:
13     ShowFile(QWidget *parent=0,const char *name=0);
14 private:
15     QLabel *filelabel;
16     QString filename;
17 private slots:
18     void popupOpen();
19     void popupSave();
20     void popupDirectory();
21     void popupFilter();
22 };
23
24 #endif
```

머리부파일은 ShowFile클라스를 선언한다. 클라스는 현재 파일 또는 등록부의 이름과 그것을 현시하는데 사용된 QLabel창문부품을 포함한다. 4개 처리부메쏘드는 각각 다른 방식으로 QFileDialog를 펼치는 단추에 련결된다. 기본창문은 표식자와 단추들을 가지고있으며 그림 5-6에 보여준다.
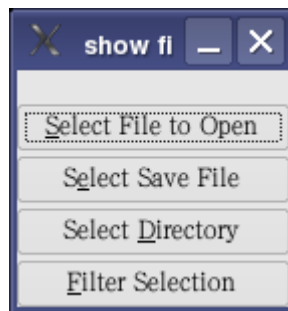
그림 5-6. 완전경로이름과 그것을 선택하는 4가지 방법

ShowFile

```
1 /* showfile.cpp */
2 #include <kapplication.h>
3 #include <qpushbutton.h>
4 #include <qlayout.h>
5 #include <qfiledialog.h>
6 #include "showfile.h"
7
```

```
8 int main(int argc, char **argv)
9 {
10     KApplication app(argc, argv,  "showfile");
11     ShowFile showfile;
12     showfile.show();
13     app.setMainWidget(&showfile);
14     return(app.exec());
15 }
16 ShowFile::ShowFile(QWidget *parent,const char *name)
17     : QWidget(parent,name)
18 {
19     QPushButton *button;
20     QVBoxLayout *box = new QVBoxLayout(this,0,3);
21
22     filelabel = new QLabel("",this);
23     filelabel->setAlignment(Qt::AlignHCenter);
24     box->addWidget(filelabel);
25
26     button = new QPushButton("Select File to Open",this);
27     box->addWidget(button);
28     connect(button,SIGNAL(clicked()),
29             this,SLOT(popupOpen()));
30
31     button = new QPushButton("Select Save File",this);
32     box->addWidget(button);
33     connect(button,SIGNAL(clicked()),
34             this,SLOT(popupSave()));
35
36     button = new QPushButton("Select Directory",this);
37     box->addWidget(button);
38     connect(button,SIGNAL(clicked()),
39             this,SLOT(popupDirectory()));
40
41     button = new QPushButton("Filter Selection",this);
42     box->addWidget(button);
```

```
43    connect(button,SIGNAL(clicked()),
44            this,SLOT(popupFilter()));
45
46    resize(10,10);
47    box->activate();
48 }
49 void ShowFile::popupOpen()
50 {
51    QString name = QFileDialog::getOpenFileName("",
52            NULL,this);
53    if(!name.isEmpty()) {
54            filename = name;
55            filelabel->setText(filename);
56    }
57 }
58 void ShowFile::popupSave()
59 {
60    QString name = QFileDialog::getSaveFileName(filename,
61            NULL,this);
62    if(!name.isEmpty()) {
63            filename = name;
64            filelabel->setText(filename);
65    }
66 }
67 void ShowFile::popupDirectory()
68 {
69    QString name = QFileDialog::getExistingDirectory();
70    if(!name.isEmpty()) {
71            filename = name;
72            filelabel->setText(filename);
73    }
74 }
75 void ShowFile::popupFilter()
76 {
77    QString filter =
```

```
78          "All (*)\n"
79          "C Source (*.c *.cc *.cpp *.cxx)\n"
80          "C Header (*.h)\n"
81          "Text (*.txt)\n"
82          "HTML (*.html *.shtml *.HTML *.htm)  ";
83     QString name = QFileDialog::getOpenFileName("",
84          filter,this);
85     if(!name.isEmpty()) {
86          filename = name;
87          filelabel->setText(filename);
88     }
89 }
```

8~15행은 프로그람의 기본함수로서 응용프로그람을 초기화한 다음 그림 5-6에 보여주는 ShowFile창문을 만들고 현시한다. 16행에서 시작하는 구성자는 수직칸안에 표식자와 4개의 단추를 포함하는 창문부품이다. 표식자는 최근에 선택한 파일 또는 등록부를 현시하는데 사용된다. 단추들은 각각 QFileDialog객체를 만들고 입력결과를 얻는 처리부에 련결된다. 4가지 조작은 각각 같은 현시형식을 사용하지만 제목띠제목과 단추조작은 좀 다르다. 그림 5-7은 QFileDialog창문의 배치를 보여준다.
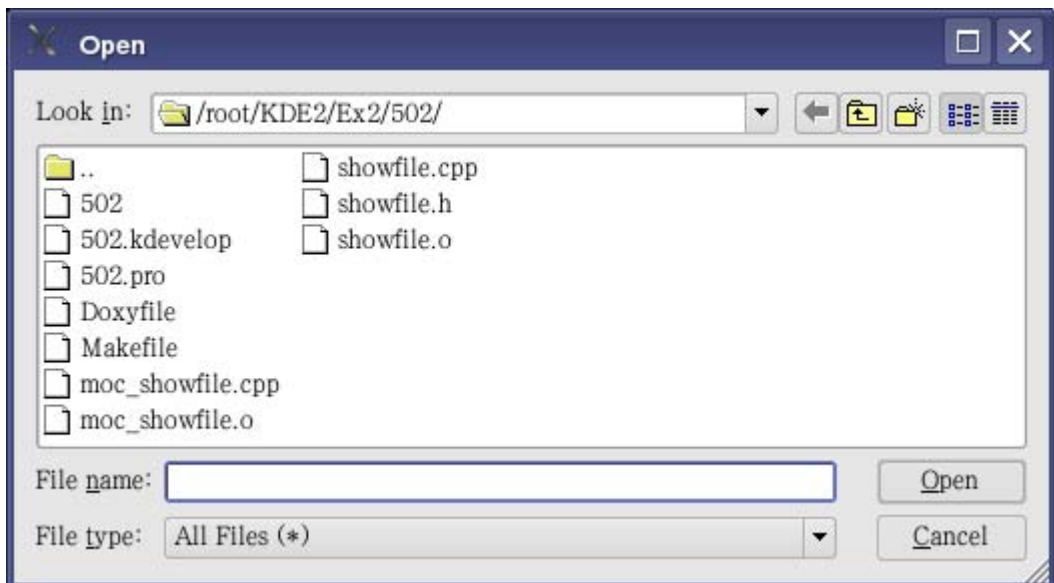
그림 5-7. 파일과 등록부들을 현시하는 QFileDialog

49행에서 시작하는 popupOpen()처리부는 사용자가 현존파일의 이름을 선택할수 있는 QFileDialog창문을 연다. 51행에서 정적함수 getOpenFileName()호출은 대화칸을 만든다. 첫째 인수(이 실례에서 빈 문자렬)는 사용자에게 제안하려는 파일의 완전경로이름이다. 어떤 파일

이름도 지정되지 않으면 대화칸은 이 응용프로그람에 의해서 최근에 호출된 등록부나 현재 등록부에 기초하여 열린다. 파일이 창문에서 선택될 때마다 단추는 본문을 OK로 바꾸고 사용자가 파일을 선택할수 있게 한다.

58행에서 시작하는 popupSave()처리부는 QFileDialog를 열고 사용자가 현존파일을 선택하거나 존재하지 않은 파일의 이름을 입력할수 있게 한다. 60행에서 getSaveFileName()호출은 대화칸을 만든다. 구성자에 넘기는 첫 인수는 마지막으로 얻어진 파일의 이름이고 이것은 창문에서 현재파일로 된다.

67행의 popupDirectory()처리부는 QFileDialog를 사용하여 등록부을 선택한다. 이 메쏘드에서 대화칸은 오직 선택된 등록부의 이름만 돌려준다. 사용자가 파일이름을 선택하면 파일이름은 생략되고 등록부경로만 돌려준다.

75행의 popupFilter()처리부는 그림 5-8에서 보여주는 파일이름려과기그룹을 지정한다. 려과기들은 창문에 표시할 파일이름들을 선택하는데 사용되는 파일이름확장자들의 집합이다. 려과기들은 모두 77행에서 시작하는 하나의 QString으로서 정의된다. 파일확장자들은 부류별로 조직되며 그림 5-8에서 보여주는 "C Source"와 같이 표시된다. 범주를 정의할 때 범주이름뒤에 괄호안에 넣은 유효파일확장자들을 쓴다. 괄호안의확장자들은 공백으로 구분한다. 서로 다른 부류들을 행바꾸기기호 '\n'에 의해 구별하거나 한쌍의 반두점(";;")에 의해 구별한다.
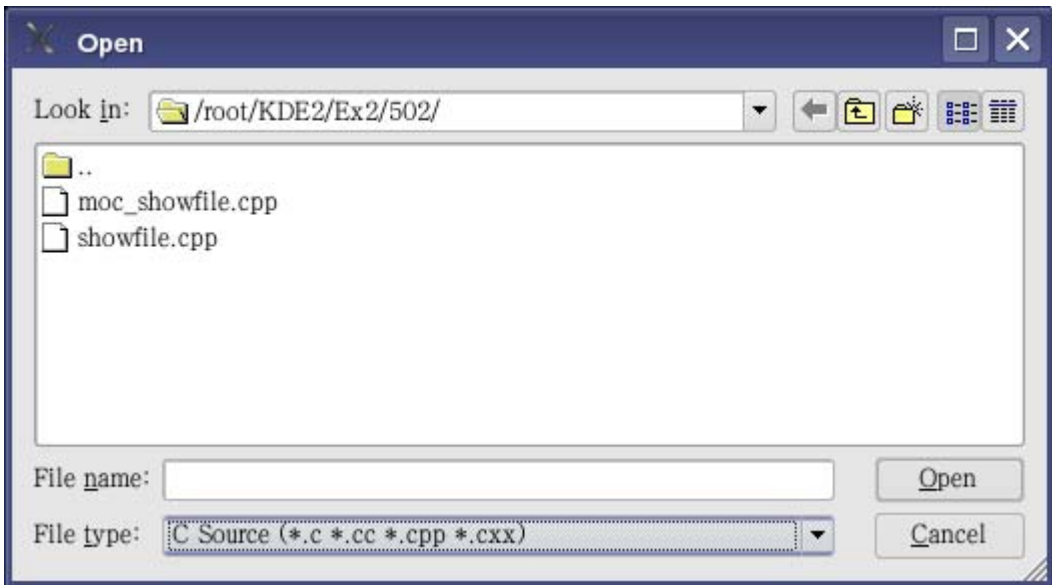
그림 5-8. 표시할 파일들을 지정하는 려과기

알아두기: 거의 모든 경우와 이 실례의 려과기들은 특정한 확장자를 가진 파일들을 선택하는데 사용된다. 그러나 려과기는 실제로 정규식이고 다른 형식에서 사용될수 있다. 실례로 려과기 sh*는 sh로 시작하는 파일들만 렬거하도록 제한한다.

그림 5-7과 5-8은 파일과 등록부이름의 배렬을 보여준다. 등록부들은 황갈색 직4각형들에 표식되고 항상 파일들앞에 렬거된다. 두 그룹에서 그것들은 ASCII값에 의해 분류되므로

대문자들은 소문자앞에 온다. 렬거된 등록부가 기호련결이면 그 그림기호는 아래의 오른쪽 구석에 작은 마크를 가진다.

> 참고: 도형파일조작을 론의하는 13장에서 설명하는것처럼 자기 그림기호들을 제공함으로써 QFileDialog창문의 모양을 사용자정의할수 있다.

제3절. QTabDialog

QTabDialog는 하나이상의 대화칸들을 하나로 묶어서 서로 절환하는데 사용할수 있는 타브들을 제공한다. 다음 프로그람은 그림 5-9에 보여주는 간단한 QTabDialog를 만든다.
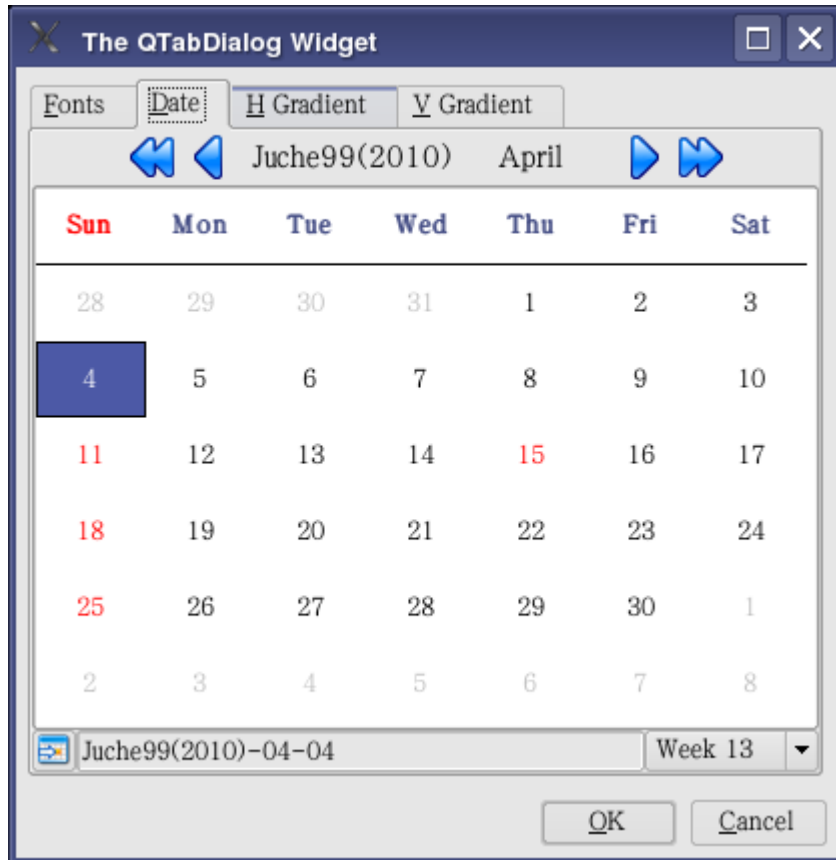
그림 5-9. 현시되는 둘째 창문부품을 가진 QTabDialog

ShowTabs머리부파일

```
1 /* showtabs.h */
2 #ifndef SHOWTABS_H
3 #define SHOWTABS_H
4
5 #include <qwidget.h>
6
7 class ShowTabs: public QWidget
```

```
8 {
9     Q_OBJECT
10 public:
11     ShowTabs(QWidget *parent=0,const char *name=0);
12 private slots:
13     void slotTab();
14 };
15
16 #endif
```

ShowTabs

```
1 /* showtabs.cpp */
2 #include <kapplication.h>
3 #include <qpushbutton.h>
4 #include <qtabdialog.h>
5 #include <qlayout.h>
6 #include <kfontdialog.h>
7 #include <kdatepik.h>
8 #include <kselect.h>
9 #include "showtabs.h"
10
11 int main(int argc, char **argv)
12 {
13     KApplication app(argc, argv,  "showtabs");
14     ShowTabs showtabs;
15     showtabs.show();
16     app.setMainWidget(&showtabs);
17     return(app.exec());
18 }
19
20 ShowTabs::ShowTabs(QWidget *parent,const char *name)
21     : QWidget(parent,name)
22 {
23     QPushButton *button;
24     QVBoxLayout *box = new QVBoxLayout(this,12);
25
```

```
26    button = new QPushButton("Show Tab Dialog",this);
27    box->addWidget(button);
28    connect(button,SIGNAL(clicked()),
29            this,SLOT(slotTab()));
30
31    resize(10,10);
32    box->activate();
33 }
34 void ShowTabs::slotTab()
35 {
36    QTabDialog *tab = new QTabDialog(this,  "tabdial",TRUE);
37    tab->setCaption("The QTabDialog Widget");
38    tab->setCancelButton();
39
40    QWidget *fonts = new KFontChooser(this,  "fonts");
41    tab->addTab(fonts,  "Fonts");
42
43    QWidget *date = new KDatePicker(this);
44    tab->addTab(date,  "Date");
45
46    QWidget *hgradient = new KGradientSelector(
47            KSelector::Horizontal,this);
48    tab->addTab(hgradient,  "H Gradient");
49
50    QWidget *vgradient = new KGradientSelector(
51            KSelector::Vertical,this);
52    tab->addTab(vgradient,  "V Gradient");
53
54    tab->show();
55 }
```

ShowTabs클라스는 오직 QTabDialog를 펼치는데 사용되므로 그 선언에 포함하는것은 구성자와 대화칸을 펼치는 처리부메쏘드뿐이다.

11행에서 시작하는 기본함수는 ShowTabs창문부품을 만들어서 기본창문창문부품으로서 설치한다.

20행에서 시작하는 ShowTabs구성자는 하나의 누름단추를 가지는 용기를 만든다. 28행에

서 단추는 slotTab()라는 처리부메쏘드에 련결된 clicked()를 가지고있다.

34행에서 시작하는 처리부메쏘드 slotTab()는 QTabDialog를 만들고 현시한다. 대화칸은 36행에서 만들어진다. 첫째 인수는 대화칸의 부모창문부품이고 둘째는 거기에 할당된 이름이며 세번째는 QTabDialog가 이행금지라는것을 지정한다. 기정은 이행허용대화칸이다.

37행에서 setCaption()호출은 대화칸창문의 제목띠에 현시하려는 제목을 설정한다. setCancelButton()호출은 Cancel단추가 대화칸의 부분으로 포함한다. 대화칸은 4개의 단추를 가질수 있다.

기정적으로 OK단추가 항상 있으나 자기가 좋아하는 다른 단추들을 지정할 필요가 있다. setDefaultButton(), setHelpButton() 및 setApplyButton()호출에 의해 다른 단추들을 포함할수 있다. 단추들을 포함하는 메쏘드들은 38행과 같이 인수없이 호출되거나 단추본문을 지정하는(setOkButton()을 포함하여) 문자렬과 함께 호출될수 있다. 단추로부터 신호들을 받아들이기 위하여 applyButtonPressed(), cancelButtonPressed(), defaultButtonPressed() 및 helpButtonPressed()에 처리부들을 련결해야 한다.

40～52행은 QTabDialog에 포함되는 4페지(보통 타브페지)를 만든다. 코드를 간단히 하기 위하여 표준KDE창문부품들중 4개를 사용하며 그것들에는 사용자가 입력한 자료를 얻는데 필요한 기능이 없다. 자체로 창문부품들을 구축할수 있다. 보통 4개의 창문부품으로부터 정보를 읽어들이는 OK와 Apply단추에 련결된 처리부가 있다.

대화칸의 크기는 그것이 포함하는 창문부품들의 크기에 의해 결정된다. 대화칸은 제일 큰 창문부품에 필요되는 높이와 너비로 표시되며 작은 창문부품들은 수직과 수평으로 중심에 배치된다. 타브를 리용하여 창문부품들을 서로 절환하는 대화칸이 점차 늘어나고있다. 또한 자료입력항목들이 차있는 페지를 현시하지 않고 사용자에게 필요한 모든 항목들을 현시한다.

제4절. QProgressDialog

가끔 프로그람은 몇초 또는 몇분 걸리는 일을 수행하여야 한다. 시간지연이 짧으면 유표를 시계화상으로 변경하여 《프로그람이 실행중에 있으니 잠간 기다리십시오》라고 전달할수 있다. 지연이 길면(실례로 15초이상) 무슨 일을 하고 있는가에 대한 정보를 사용자에게 주는것이 편리하다. QProgressDialog는 과제완성률을 현시하는데 사용되며 지속시간이 충분히 길 때에만 펼치도록 설정할수 있다.

다음 프로그람은 QProgressDialog를 사용하는 2가지 방법을 보여준다. 그림 5-10은 QProgressDialog창문을 펼치는데 쓰이는 2개의 단추를 가지는 응용프로그람의 기본창문을 보여준다. 이 대화칸은 보통 자료전송, 큰 파일의 분류 혹은 시간이 걸리는 일의 진행상황을 현시하는데 쓰이며 이 프로그람의 실례들은 시계의 진행상황에 기초하여 간단히 조작한다.
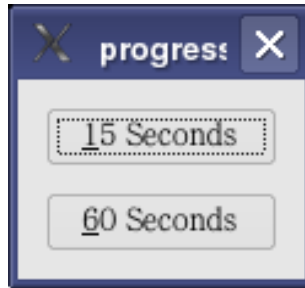
그림 5-10. 2개의 QProgressDialog창문중 하나를 기동한다

Progress머리부파일

```
1 /* progress.h */
2 #ifndef PROGRESS_H
3 #define PROGRESS_H
4
5 #include <qprogressdialog.h>
6 #include <qwidget.h>
7 #include <qtimer.h>
8
9 class Progress: public QWidget
10 {
11     Q_OBJECT
12 public:
13     Progress(QWidget *parent=0,const char *name=0);
14 private:
15     QProgressDialog *progressDialog;
16     QTimer *timer;
17 private slots:
18     void slot15();
19     void slot60();
20     void timerStep();
21 };
22
23 #endif
```

Progress클라스는 3개의 처리부와 하나의 시계를 가지고있다. 각각 15s와 60s동안 지속되는 진행상황대화칸을 기동하기 위하여 처리부메쏘드 slot15()와 slot60()를 각각 호출한다. 시계와 timerStep()처리부는 내부적으로 경과시간을 추적하는데 사용된다.

Progress

```
1 /* progress.cpp */
2 #include <unistd.h>
3 #include <kapplication.h>
4 #include <qpushbutton.h>
5 #include <qlayout.h>
6 #include "progress.h"
7
8 int main(int argc, char **argv)
9 {
10     KApplication app(argc, argv,  "progress");
11     Progress progress;
12     progress.show();
13     app.setMainWidget(&progress);
14     return(app.exec());
15 }
16
17 Progress::Progress(QWidget *parent,const char *name)
18     : QWidget(parent,name)
19 {
20     QPushButton *button;
21     QVBoxLayout *box = new QVBoxLayout(this,12);
22
23     button = new QPushButton("15 Seconds",this);
24     box->addWidget(button);
25     connect(button,SIGNAL(clicked()),
26             this,SLOT(slot15()));
27
28     button = new QPushButton("60 Seconds",this);
29     box->addWidget(button);
30     connect(button,SIGNAL(clicked()),
31             this,SLOT(slot60()));
32
33     resize(10,10);
34     box->activate();
35 }
```

```
36 void Progress::slot15()
37 {
38     int currentStep = 0;
39     int steps = 15;
40
41     progressDialog = new QProgressDialog(
42             "Fifteen seconds..",  "Cancel",
43             steps,this,  "pgrs",TRUE);
44     progressDialog->setCaption("Progress");
45     while(currentStep < steps) {
46             progressDialog->setProgress(currentStep++);
47             if(progressDialog->wasCancelled())
48                     break;
49             sleep(1);
50     }
51     progressDialog->setProgress(steps);
52     delete progressDialog;
53     progressDialog = NULL;
54 }
55 void Progress::slot60()
56 {
57     int currentStep = 0;
58     int steps = 20;
59
60     progressDialog = new QProgressDialog(this,  "prgs",TRUE);
61     progressDialog->setCaption("Progress");
62     progressDialog->setLabelText("Sixty seconds...");
63     progressDialog->setCancelButtonText("Quit");
64     progressDialog->setTotalSteps(steps);
65     progressDialog->setMinimumDuration(3000);
66
67     timer = new QTimer(this);
68     connect(timer,SIGNAL(timeout()),
69             this,SLOT(timerStep()));
70     timer->start(3000,FALSE);
```

```
71 }
72 void Progress::timerStep()
73 {
74     int currentStep;
75     int steps = 20;
76
77     if(progressDialog == NULL)
78             return;
79
80     if(progressDialog->wasCancelled()) {
81             delete timer;
82             delete progressDialog;
83             progressDialog = NULL;
84             return;
85     }
86     currentStep = progressDialog->progress();
87     if(currentStep >= steps) {
88             delete timer;
89             delete progressDialog;
90             progressDialog = NULL;
91     } else {
92             progressDialog->setProgress(currentStep + 1);
93     }
94 }
```

8행에서 시작하는 프로그람의 기본함수는 Progress객체를 만들고 그것을 제일웃준위창문으로 설치한다. 제일웃준위창문부품은 17행에서 구성자에 의해 만들어진 Progress창문부품이다. 수직칸은 2개의 누름단추를 포함한다. 단추들은 처리부 slot15()와 slot60()에 련결된다. 결과는 그림 5-15에 보여주는 창문이다.

36행에서 시작하는 처리부메쏘드 slot15()는 순환안에서 진척상황띠를 사용하는 방법을 보여준다. 진행상황률은 현재걸음수와 총걸음수의 비률에 의해 결정된다. 38행은 현재걸음(시작걸음)을 0으로 지정하고 39행은 총걸음수를 15로 정의한다. 41행의 구성자는 QProgressDialog를 만들고 총걸음수를 설정한다. 또한 제목본문과 창문부품이름을 설정하고 대화칸이 이행금지대화칸이라는것을 TRUE로 지정한다. 44행의 setCaption()호출은 대화칸제목띠의 본문을 지정한다.

알아두기: 여기에 포함된 실례들은 둘다 이행금지대화칸이지만 보통 이행허용진행상

황대화칸을 사용한다. 실례로 웹열람기가 하나 또는 두개 파일을 내리적재하고 있을 때 매개 적재는 독립적인 진행상황대화칸을 가지며 여전히 열람기의 다른 기능을 호출할수도 있다.

45행에서 시작하는 순환고리는 진척상황띠에 수행할 조작을 모의하기 위한것이다. 순환고리가 하는 일은 현재걸음계수값을 증가하고 1s동안 휴식시키는것이다. 현실세계응용프로그람에서는 다음 걸음의 값을 계산하고 그 값을 가지고 setProgress()를 호출한다. 47행에서 wasCancelled()호출은 사용자가 Cancel단추를 선택하면 TRUE를 돌려주므로 순환고리는 빨리 완료한다. 결과는 그림 5-11에 보여준 진행상황창문이다. 보통 일정한 시간이 지나면 Cancel단추를 선택하여 중지코드를 순환안에 넣고 순환밖으로 이동한다.
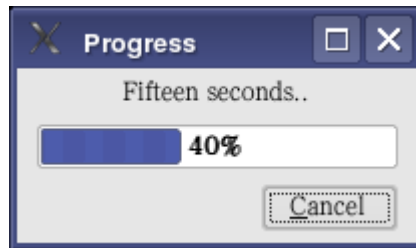
그림 5-11. 15개 걸음을 가지는 QProgressDialog

51행의 setProgress()호출은 대화칸이 적당한 때에 닫기도록 한다. QProgressDialog이행금지대화칸이 실행을 시작할 때 우선 유표가 시계기호로 변한다. 그다음 대화칸이 펼쳐지기전에 잠간 기다린다. 현재값이 증가될 때마다 현시는 갱신된다. 마지막 걸음에 이르면 원래의 유표를 되살리여 펼친다. 51행의 setProgress()호출은 순환이 최대걸음값에 이르지 못한채 완료할 때 필요하다.

55행에서 시작하는 slot60()라는 처리부메쏘드는 그림 5-12에 보여준것처럼 전혀 다른 메쏘드를 사용하여 QProgressDialog를 만들고 갱신한다. 57행과 58행은 시작값을 0으로, 걸음수를 20으로 설정한다. 대화칸은 60행에서 이전 실례보다 더 간단한 구성자에 의하여 만들어진다. 61～65행의 메쏘드호출은 창문제목본문을 설정하고 대화칸에 나타나는 표식자의 본문을 지정하며 Cancel단추의 본문을 바꾸고 그리고 총걸음수를 완료로 설정한다.
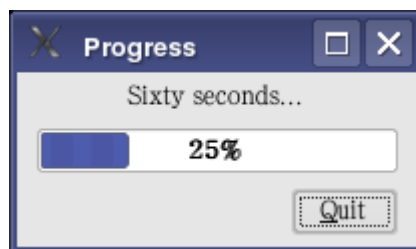
그림 5-12. 신호를 사용하는 QProgressDialog

65행의 setMinimumDuration()호출은 지연시간을 3000ms(3s)로 설정한다. 이것은 대화칸을 펼치기전에 기다리는 시간량이다.

3000ms로 설정하면 유표가 시계로 변화되고 3s지날 때까지 아무일도 하지 않는다. 이리하여 대화칸은 3s보다 적게 걸리는 과제에는 절대로 나타나지 않는다.

67～70행은 시계를 만들고 실행을 시작한다. 시계의 timeout()신호는 이 클라스의 timerStep()에 련결된다. 70행에서 start()호출은 시계가 3s에 끝나도록 초기화한다.둘째 파라메터를 FALSE로 설정하면 이것은 한번만 현시되는 시계가 아니라는것을 의미하며 정지될 때까지 3s마다 현시되면서 계속 실행된다. 둘째 파라메터를 TRUE로 설정하면 시계는 오직 한번만 현시된다. 실제로 실행중에 있는 시계를 얻는 하나의 방법은 대체로 다른 방법과 비슷하다. 그것이 계속 실행되고 있다면 최후에는 없어져야 한다. 그것이 한번만 현시되는 시계이면 그것이 현시될 때마다 그것을 재기동시킬수 있다.

72행의 timerStep()처리부는 시계가 발사할 때마다 실행한다. 사용자가 조작을 정지시키기 위하여 Cancel단추를 선택하면 80행의 wasCancelled()호출은 TRUE로 된다. 86행의 progress()호출은 현재걸음값을 얻으며 87행에서 그것이 끝에 도달했다는것을 발견하면 시계와 대화칸은 둘다 삭제된다. 그것이 여전히 진척중에 있다면 92행의 setProgress()호출은 현재걸음수를 증가시킨다.

progressDialog지적자는 대화칸이 해체될 때 NULL로 설정되고 77행에서 그것이 NULL인가를 시험한다. 이 시험은 QTimer가 일하는 방법이므로 필요하다. 동기를 얻기 위하여 QTimer는 사건대기렬에 특수사건을 삽입하며 사건대기렬은 응용프로그람과 비동기이므로 QTimer는 QProgressDialog이 삭제될 때 시간사건이 입력대기렬에 아직 있을수 있다. 이것은 처리부 timerStep()가 한번이상 실행된다는것을 의미한다.

요 약

이 장은 KDE와 Qt대화칸들중 일부를 사용하는 방법을 보여주었다. 미리 정의된 대화칸은 더 많으며 이 대화칸들은 많은 항목들을 가지고있다. 이 장은 다음과 같은 대화칸의 사용방법을 설명하였다.

· 자기 응용프로그람이 KDE체계의 부분으로 인정될수 있는 충분히 표준인 About창문을 만든다.

· 파일과 등록부들을 제시하고 사용자가 선택하도록 한다.

· 여러개의 창문부품들을 한곳에 배치하고 창문부품들을 서로 절환하기 위한 타브대화칸을 사용자에게 제공한다.

· 프로그람이 무엇인가 수행하고 있다는것을 사용자에게 보증하기 위하여 진척상황띠를 현시한다.

이 장에서 설명하지 않은 대화칸들은 관련한 장들에서 설명한다. 그러나 대화칸은 펼칠수 있는 유일한것이 아니다. 다음 장은 차림표와 도구띠를 만들고 현시하는 과정을 시험한다.

제6장. 차림표와 도구띠

학습내용

차림표띠를 가지는 기본응용프로그람창문의 제공,
창문부품안으로부터 차림표의 펼치기,
기본응용프로그람창문안에 하나이상의 도구띠현시,
기본창문우의 상태띠안의 본문갱신방법.

이 장은 프로그람의 요소를 호출하고 프로그람에서 무엇을 수행하는가에 대하여 사용자에게 통보하는데 쓰이는 창문부품들로 응용프로그람의 기본창문을 꾸미는 방법에 대하여 학습한다. 이것은 대체로 응용프로그람의 기본창문에서 수행되며 KMainWindow라는 특수한 제일웃준위창문클라스가 있다. 그것은 차림표띠, 도구띠 그리고 상태띠를 관리하는데 필요한 모든것을 포함하고 있다. 필요할 때 KMainWindow는 이 항목들을 구성하고 자기 응용프로그람의 기본창문으로 사용하려는 창문부품을 삽입할 곳을 제공하며 그다음 포함되여있는 여러 부분들과의 사용자교제를 관리한다. 이 창문부품들을 모두 만들고 관리할수 있지만 KmainWindow가 그것을 수행하게 하는것이 더 간단하다.

제1절. KMainWindow

KMainWindow클라스는 응용프로그람의 제일웃준위창문에 리상적이도록 설계된 편리한 기능들을 결합한 창문부품이다. 그것은 기본창문부품(또한 보기창문부품라고 한다)을 보관할뿐아니라 차림표띠, 상태띠, 그리고 하나이상의 도구띠를 만들고 관리하는 기본기구를 조종한다.

알아두기: KMainWindow가 닫길 때에는 내부적으로 할당된 기억기를 해방하므로 KMainWindow는 항상 새로운 지령으로 창조되여야 한다. 그것이 대역객체로 또는 탄창에 정의된다면 프로그람은 기억기를 해방하려고 시도할 때 중단된다. (KDE2까지 KTMainWindow으로 되여있던것이 KDE 3에서는 KMainWindow로 변경되였다.)

다음 실례프로그람은 그림 6-1에 보여준것처럼 제일웃준위창문부품을 둘러싸는 하나의 차림표, 도구띠, 그리고 상태띠를 가지고 있는 제일웃준위창문을 창조하는데 KMainWindow를 사용한다.

그림 6-1. KMainWindow을 사용한 응용프로그람의 기본창문창조와 조종

SimpleMain머리부파일

```
1 /* simplemain.h */
2 #ifndef SIMPLEMAIN_H
3 #define SIMPLEMAIN_H
4
5 #include <kmainwindow.h>
6 #include <kmenubar.h>
7 #include <ktoolbar.h>
8 #include <kstatusbar.h>
9
10 class SimpleMain: public KMainWindow
11 {
12     Q_OBJECT
13 public:
14     SimpleMain();
15 private slots:
16     void slotExit();
17     bool queryClose();
18 private:
19     void createMainWidget();
20     void createMenu();
21     void createStatusBar();
22     void createToolBar();
23 };
24
25 #endif
```

5～8행은 KMainWindow클라스와 기본창문의 부분으로 포함될 3개 클라스를 선언하는 머리부파일들을 포함한다. 10행에서 SimpleMain클라스선언은 KMainWindow를 기초클라스로 지정한다. KMainWindow의 실례를 직접 구성할수 있으며 한편 그것을 기초클라스로 사용하여 훨씬 더 많은 유연성을 제공한다.

SimpleMain

```
1 /* simplemain.cpp */
2 #include <kapplication.h>
3 #include <kcmdlineargs.h>
4 #include <qpushbutton.h>
```

```
5 #include "simplemain.h"
6
7 int main(int argc, char **argv)
8 {
9     KCmdLineArgs::init(argc, argv,  "simplemain",
10            "Simple Main",  "0.0");
11    KApplication app;
12    SimpleMain *simplemain = new SimpleMain();
13    simplemain->show();
14    return(app.exec());
15 }
16 SimpleMain::SimpleMain() : KMainWindow()
17 {
18    createMainWidget();
19    createMenu();
20    createStatusBar();
21    createToolBar();
22 }
23 void SimpleMain::createMainWidget()
24 {
25    QPushButton *button =
26            new QPushButton("Top Level\nWidget",this);
27    setCentralWidget(button);
28 }
29 void SimpleMain::createMenu()
30 {
31    KMenuBar *menubar = menuBar();
32    QPopupMenu *popup = new QPopupMenu();
33    popup->insertItem("E&xit",this,SLOT(slotExit()));
34    menubar->insertItem("&File",popup);
35 }
36 void SimpleMain::createStatusBar()
37 {
38    KStatusBar *status = statusBar();
39    status->insertItem("Status Bar",1);
```

```
40 }
41 void SimpleMain::createToolBar()
42 {
43     KToolBar *toolBar = ToolBar(0);
44     QPixmap pixmap("flag.png");
45     toolBar->insertButton(pixmap,5);
46 }
47 void SimpleMain::slotExit()
48 {
49     kapp->exit(0);
50 }
51 bool SimpleMain::queryClose()
52 {
53     return(TRUE);
54 }
```

프로그람의 기본함수는 7∼15행에서 정의된다. 11행은 KApplication객체를 만들고 KDE를 초기화한다. 응용프로그람의 기본창문은 12행에서 만들어진다. app객체와 SimpleMain객체(즉 KMainWindow객체)사이의 관계는 내부적으로 확립되므로 두 객체를 련결할 필요는 없다. 이것이 바로 응용프로그람에 유일한 KMainWindow객체가 있는 리유이다. 13행에서 show()호출은 기본창문을 현시하게 하므로 프로그람은 14행에서 exec()를 호출함으로써 지령순환고리에 들어간다.

구성자는 16행에서 정의된다. 구성자는 4개 메쏘드를 호출하여 기본창문현시의 4가지 요소를 만든다. 23행에서 메쏘드 createMainWidget()는 응용프로그람의 기본창문으로 되는 창문부품을 만든다. 이 실례에서 기본창문은 간단히 25행에서 창조한 QPushButton이다. 27행에서 setCentralWidget()호출은 기본창문 즉 보기로서 그 창문부품을 삽입한다.

차림표띠는 29행에서 시작하는 메쏘드 createMenu()에서 구성되고 차림표띠는 31행에서 메쏘드 menuBar()호출에 의해 만들어진다. 이 메쏘드는 여러번 호출될수 있고 KMainWindow에는 하나의 차림표띠만 있으므로 늘 같은 KMenuBar지적자를 돌려준다. menuBar()의 첫 호출은 차림표가 이미 존재하지 않을 때에만 새로 만든다. 만약 필요하다면 자체의 차림표를 만들고 다음과 같이 KMainWindow에 그것을 삽입할수 있다.

```
KMenuBar *myMenuBar = new KMenuBar();
setMenu(myMenuBar);
```

차림표띠를 만들기 위하여 32행에서 QPopupMenu객체를 만든다. 이 객체를 차림표항목들의 렬이 튀여나오는 용기로서 동작시키기 위하여 KMenuBar에 삽입할수 있다. 실례의 33행에서 insertItem()호출에 의해 하나의 단추를 삽입한다. 단추는 “E&xit”로 표식되며 처리부

메쏘드 slotExit()를 호출한다. 34행에서 insertItem()호출은 차림표띠에 튀여나오기차림표를 삽입한다.

그림 6-2에 보여준것처럼 사용자는 창문의 측면으로 도구띠를 끌고가서 도구띠를 수직방향으로 변환한다. 또한 창문의 오른쪽 혹은 바닥에 도구띠를 배치할수 있다.
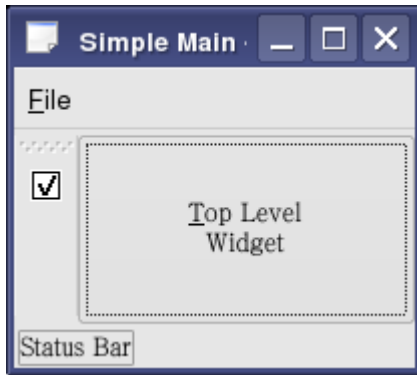
그림 6-2. 차림표띠와 도구띠의 위치설정

36행의 메쏘드 createStatusBar()는 statusBar()를 호출하여 단일행의 본문을 현시할수 있는 창문부품을 만들고 설치한다. 프로그람은 상태띠의 본문을 호출하므로 그것은 계속 갱신할수 있다. 도구띠와 차림표띠와는 달리 상태띠는 다른 장소로 움직일수 없다.

41행의 메쏘드 createToolBar()는 43행의 toolbar()를 호출하여 도구띠를 만든다. 응용프로그람은 필요한만큼 도구띠를 가질수 있으므로 ID번호를 인수로서 요구한다. ID를 가진 도구띠가 없으면 그 도구띠를 만들고 돌아온다. 같은 ID번호를 사용하면 늘 같은 도구띠를 돌려준다. 이 실례에서 하나의 픽스매프가 하나의 도구띠단추를 만드는데 사용된다.

도구띠는 그 부모창문밖으로 이동될수 있다. 그림 6-3은 도구띠의 왼쪽끝에 있는 손잡이를 사용하여 그 부모창문으로부터 도구띠를 분리하여 자기의 독립창문으로 설정한 결과를 보여준다.

그림 6-3. 류동차림표와 도구띠

47행의 slotExit()라는 메쏘드는 사용자가 차림표로부터 Exit를 선택할 때마다 호출된다. 여기서 일부 삭제하고 현재 보관하지 않은 자료를 보관할 필요가 있으나 이 실례는 간단히 응용프로그람의 exit()를 호출한다. kapp라는 대역변수는 늘 KApplication객체의 지적자를 포함한다.

51행의 queryClose()라는 처리부메쏘드는 응용프로그람의 틀의 오른쪽웃구석에 있는 그림기호 X가 선택될 때 호출된다. 이 처리부가 TRUE를 돌려주면 응용프로그람은 즉시 닫기고 FALSE를 돌려주면 어떤 작용도 없다(X단추로부터 오는 신호는 무시된다).

이것은 응용프로그람의 골격이다. KMainWindow가 대부분의 일을 수행하므로 제일웃준위창문은 단 몇행의 코드로 아주 복잡한 동작을 수행할수 있다. 이 장의 나머지는 차림표띠와 도구띠의 환경구성과 상태띠를 통한 정보현시방법을 자세히 설명한다.

제2절. 차림표띠

주로 차림표는 그 매개 항목에 련결된 처리부를 가진 단추들의 집합이다. 차림표의 동특성은 단추들의 특정한 묶음을 임의로 사용할수 있게 함으로써 단추호출을 간단하게 한다. 단추들을 배치하고 장식하는 방법이 여러가지 있다. 다음 실례는 그림 6-4에 보여주는 창문을 만든다. 이 차림표띠는 차림표를 만드는 방법을 설명하기 위한것이다.

그림 6-4. 웃끝에 차림표띠를 가지는 응용프로그람

머리부파일은 차림표단추가 선택될 때마다 호출되는 처리부들의 선언을 포함하지만 한 상태에서 다른 상태에로 절환되는 차림표단추들의 상태를 추적하는데 필요한 비공개자료가 있다. MenuMain클라스는 KMainWindow클라스를 계승하므로 이미 차림표를 현시하고 관리할 능력을 가진다.

MenuMain머리부파일

```
1 /* menumain.h */
2 #ifndef MENUMAIN_H
3 #define MENUMAIN_H
4
5 #include <kmainwindow.h>
6 #include <kmenubar.h>
7 #include <ktoolbar.h>
```

```
8 #include <kstatusbar.h>
9
10 class MenuMain: public KMainWindow
11 {
12     Q_OBJECT
13 public:
14     MenuMain();
15 private:
16     QPopupMenu *checkPopup;
17     int enableColorsID;
18     int enableGraphicsID;
19 private slots:
20     void slotExit();
21     bool queryClose();
22     void slotNew();
23     void slotSave();
24     void slotSaveAs();
25     void slotClose();
26     void slotLogo();
27     void slotSub();
28     void slotEnableColors();
29     void slotEnableGraphics();
30 private:
31     void createMainWidget();
32     void createMenu();
33 };
34
35 #endif
```

MenuMain

```
1 /* menumain.cpp */
2 #include <kapplication.h>
3 #include <khelpmenu.h>
4 #include <kcmdlineargs.h>
5 #include <qpushbutton.h>
6 #include <qwhatsthis.h>
```

```
7 #include "menumain.h"
8
9 int main(int argc, char **argv)
10 {
11     KCmdLineArgs::init(argc, argv,  "menumain",
12             "Menu Main",  "0.0");
13     KApplication app;
14     MenuMain *menumain = new MenuMain();
15     menumain->show();
16     return(app.exec());
17 }
18 MenuMain::MenuMain() : KMainWindow()
19 {
20     createMainWidget();
21     createMenu();
22 }
23 void MenuMain::createMainWidget()
24 {
25     QPushButton *button =
26             new QPushButton("Top Level\nWidget",this);
27     QWhatsThis::add(button,
28             "Button\n\n"
29             "This button is used as the top\n"
30             "level widget for this example. It\n"
31             "is very safe to click the button\n"
32             "because it doesn’t do anything.\n");
33     setCentralWidget(button);
34 }
35 void MenuMain::createMenu()
36 {
37     QPopupMenu *popup;
38     QPopupMenu *popup2;
39     QPixmap pixmap;
40     KMenuBar *menubar = menuBar();
41
```

```
42    popup = new QPopupMenu();
43    popup->insertItem("&New",this,
44            SLOT(slotNew()),ALT+Key_N);
45    popup->insertItem("&Save",this,
46            SLOT(slotSave()),CTRL+Key_S);
47    popup->insertItem("Save As",this,
48            SLOT(slotSaveAs()),CTRL+SHIFT+Key_S);
49    pixmap.load("flag.png");
50    QIconSet iconset(pixmap);
51    popup->insertItem(iconset,  "Close",this,
52            SLOT(slotClose()));
53    popup->insertSeparator();
54    popup->insertItem("Exit",this,
55            SLOT(slotExit()),ALT+Key_X);
56    menubar->insertItem("&File",popup);
57
58    checkPopup = new QPopupMenu();
59    checkPopup->setCheckable(TRUE);
60    enableColorsID = checkPopup->insertItem(
61            "Enable Colors",this,SLOT(slotEnableColors()));
62    checkPopup->setItemChecked(enableColorsID,TRUE);
63    enableGraphicsID = checkPopup->insertItem(
64            "Enable Graphics",this,
65            SLOT(slotEnableGraphics()));
66    checkPopup->setItemChecked(enableGraphicsID,FALSE);
67    menubar->insertItem("&Toggles",checkPopup);
68
69    popup = new QPopupMenu();
70    pixmap.load("tinylogo.png");
71    popup->insertItem(pixmap,this,SLOT(slotLogo()));
72    pixmap.load("qtlogo.png");
73    popup->insertItem(pixmap,this,SLOT(slotLogo()));
74    menubar->insertItem("&Pixmaps",popup);
75
76    popup = new QPopupMenu();
```

```
77     popup2 = new QPopupMenu();
78     popup2->insertItem("Horizontal",this,SLOT(slotSub()));
79     popup2->insertItem("Vertical",this,SLOT(slotSub()));
80     popup->insertItem("Orientation...  ",popup2);
81     menubar->insertItem("Submenu",popup);
82
83     KHelpMenu *help = new KHelpMenu(this,
84             "Text that will appear in\n"
85             "a very simple About box");
86     popup = help->menu();
87     menubar->insertItem("&Help",popup);
88 }
89 void MenuMain::slotExit()
90 {
91     kapp->exit(0);
92 }
93 bool MenuMain::queryClose()
94 {
95     return(TRUE);
96 }
97 void MenuMain::slotEnableColors()
98 {
99     if(checkPopup->isItemChecked(enableColorsID))
100            checkPopup->setItemChecked(enableColorsID,FALSE);
101    else
102            checkPopup->setItemChecked(enableColorsID,TRUE);
103 }
104 void MenuMain::slotEnableGraphics()
105 {
106    if(checkPopup->isItemChecked(enableGraphicsID))
107            checkPopup->setItemChecked(enableGraphicsID,FALSE);
108    else
109            checkPopup->setItemChecked(enableGraphicsID,TRUE);
110 }
111 void MenuMain::slotNew() {}
```

```
112 void MenuMain::slotSave() {}
113 void MenuMain::slotSaveAs() {}
114 void MenuMain::slotClose() {}
115 void MenuMain::slotLogo() {}
116 void MenuMain::slotSub() {}
```

10행에서 시작하는 프로그람의 기본함수는 MenuMain객체를 만들고 그것을 현시하며 exec()를 호출하고 입력을 기다린다. 15행에서 시작되는 MenuMain의 구성자는 createMainWidget()를 호출하여 창문부품이 기본창문으로 작용하도록 하고 그 다음 createMenu()를 호출하여 창문꼭대기에 차림표띠를 추가한다.

23행에서 시작하는 메쏘드 createMainWidget()는 누름단추를 만들고 그것을 응용프로그람의 기본창문으로 설치한다. 단추는 33행의 setCentralWidget()호출에 의해 설치된다. 27행의 QWhatsThis::add()호출은 단추와 설명본문을 련결하여 Help차림표로부터 본문을 현시할수 있도록 하기 위한것이다.

35행의 메쏘드 createMenu()는 차림표띠와 그 항목들을 모두 만든다. KMainWindow창문부품에 하나의 차림표띠가 있고 37행에서 그 주소를 얻어서 menubar지적자에 보관한다. 실제로 차림표띠는 menuBar()를 호출할 때까지 존재하지 않지만 다음번 menuBar()호출은 같은 차림표띠지적자를 돌려준다.

차림표띠에 나타나는 매개 단추는 하나의 QPopupMenu객체를 표시한다. 첫째 단추는 42행에서 만들어지고 56행에서 차림표띠에 추가된다. 42～56행에서 많은 항목들이 튀여나오기 차림표에 삽입되여 그림 6-5에 보여주는것과 같은 차림표가 생긴다.
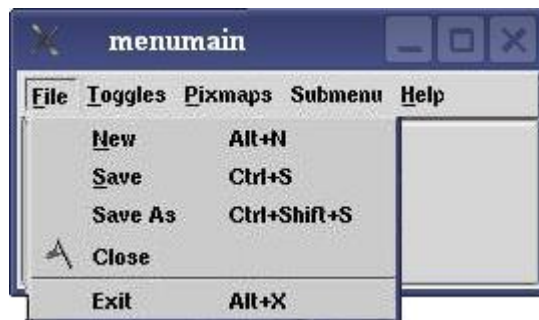
그림 6-5. 그림기호와 지름건을 가진 차림표

56행의 insertItem()호출은 차림표이름이 "File"이고 지름건이 Alt-F라는것을 지정한다. 즉 간단히 "File"대신에 "&File"라고 표식자를 쓰면 그것이 현시될 때 밑선이 있는 글자로 표시되며 Alt-F건결합을 누르면 마우스로 차림표를 선택한것과 같아진다. 올리와 내리화살표건으로 차림표항목을 찾고 Return 또는 Enter건으로 선택한다.

"New"로 표식된 항목은 43행에서 insertItem()호출에 의해 만들어진다. 지름건은 상수값 ALT+Key_N에 의해 지정된다. 이것은 차림표항목을 선택하기 위한 지름길이며 응용프로그람이 건반초점을 가질 때마다 ALT-N을 누르면 마우스로 New항목을 선택한것과 같은 결과를

산출한다. 지름건은 그림 6-5과 같이 오른쪽에 나타난다. 문자 N앞의 &는 밑선이 그려지게 하지만 차림표띠와 달리 이것은 자동적으로 지름건을 할당하지 않는다. 밑줄을 그리는가 아닌가는 선택할수 있다. "Save As"선택은 지름건을 가지만 밑줄문자를 가지지 않는다.

지름건은 건과 그 수식자들을 지정하는 특수값에 의해서 정의된다. 3개의 수식자 즉 ALT, CTRL 및 SHIFT를 사용할수 있다. 건만 사용하거나 수식건 한개, 두개, 또는 3개를 결합하여 사용할수 있다. 실례로 47행에서 정의된 "Save As"항목은 CTRL과 SHIFT수식자를 모두 사용한다. 많은 건을 지름건으로 사용할수 있다. 다음은 일반적으로 사용하는 건들을 보여준다.

정의된 건은 1개 건반에 있는것보다 더 많다. 그러나 여기에 제시하는 대부분의 건은 아주 쓸모있다. 자기가 사용하려는 특수건이 있다면 qnamespace.h라는 Qt머리부파일에서 230개이상의 건을 찾을수 있다.

Key_0 through Key_9	Key_Down	Key_Period
Key_Apostrophe	Key_End	Key_Plus
Key_Asterisk	Key_Enter	Key_Print
Key_A through Key_Z	Key_Equal	Key_QuoteDbl
Key_Backslash	Key_Escape	Key_Return
Key_BackSpace	Key_F1 through Key_F35	Key_Right
Key_Backspace	Key_Home	Key_ScrollLock
Key_BraceLeft	Key_Insert	Key_Semicolon
Key_BraceRight	Key_Left	Key_Slash
Key_BracketLeft	Key_Minus	Key_Space
Key_BracketRight	Key_Next	Key_SysReq
Key_CapsLock	Key_NumLock	Key_Tab
Key_Colon	Key_PageDown	Key_Underscore
Key_Comma	Key_PageUp	Key_Up
Key_Delete		

49～52행에 정의된 Close차림표항목은 그림 6-5에서 그 왼쪽에 그림기호와 함께 현시된다. 그러기 위하여 첫째로 화상을 포함하는 픽스매프를 만들어야 한다. 이것은 49행에서 flag.png이라는 국부디스크파일로부터 자료를 적재하여 수행한다. 둘째로 50행에서처럼 픽스매프로부터 QIconSet객체를 만들어야 한다. 차림표항목 자체는 48행에서 첫 인수로서 QIconSet를 넘기는 insertItem()호출에 의해 만들어지고 그림기호들을 포함하지 않는 경우에 인수들은 이전과 같다.

참고: 13장 《도형조작》은 픽스매프를 만드는 다른 방법을 설명한다.

53행은 Close와 Exit사이에 수평분리선을 삽입하여 File차림표를 완성한다. 54행과 55행은 Exit차림표성원을 만든다.

그림 6-6은 on과 off를 절환할수 있는 한쌍의 검사단추를 가지는 차림표를 표시한다. 검사표식자는 단추가 on으로 절환될 때 나타난다. 그림에서 Enable Colors단추는 on이고 Enable Graphics단추는 off이다.

그림 6-6. 절환단추를 가진 차림표

Toggles차림표는 58행에서 만들어지고 74행에서 차림표띠에 삽입된다. 59행의 setCheckable()호출은 모든 항목이 on과 off를 절환할수 있도록 튀여나오기차림표를 구성한다. 이리하여 간단히 on과 off를 절환함으로써 튀여나오기차림표의 임의의 항목을 절환항목으로 만든다.

실제로는 자동적으로 절환되지 않는다. 97～110행에서 절환단추용의 2개 처리부는 isItemCheck()를 호출하여 현재절환상태를 검사한 다음 setItemChecked()를 호출하여 그것을 다른 상태로 절환한다. 처리부메쏘드에서 단추들을 절환하는데 인수들이 필요하므로 머리부파일의 16～18행에서 2개 단추의 ID번호와 함께 튀여나오기차림표의 지적자를 보관할 필요가 있다. ID번호는 60행과 63행에서 itemInsert()로부터의 돌림값이며 튀여나오기차림표에서 특정한 항목을 호출하는 유일한 방법이다.

본문대신에 픽스매프로 차림표를 장식할수 있다. 그림 6-7은 단추들에 픽스매프를 사용하는 차림표들을 보여준다. 그 차림표들은 본문차림표단추들처럼 작업하므로 지름건을 가질수 있고 절환단추일수 있다. 그림에서 어떤 픽스매프는 다른 것보다 클수 있으므로 픽스매프의 크기에 따라 매개 차림표항목이 확장된다는것을 알수 있다.

그림 6-7. 단추에 픽스매프를 사용하는 차림표

픽스매프를 가진 튀여나오기차림표는 70～74행에서 만들어진다. 같은 QPixmap객체가 두 단추에 사용되며 이것은 insertItem()메쏘드가 자기의 국부사본을 만들기때문에 동작한다는것을 알아야 한다. 픽스매프와 본문단추사이의 유일한 차이는 71행과 73행에서 첫 인수의 형이고 QPixmap객체참고가 첫 인수이다.

보조차림표는 다른 차림표에 항목들중의 하나로서 튀여나오기차림표를 삽입하여 만들수 있다. 76∼81행은 popup2라는 이름의 두번째 튀여나오기차림표를 만들고 "Orientation…"로 표식하여 부모차림표에 삽입한다. 결과 차림표를 선택하면 그림 6-8과 같이 보인다.

그림 6-8. 보조차림표를 가진 차림표

KHelpMenu객체는 80행에서 창조되고 86행과 87행에서 차림표띠의 제일 오른쪽 성원으로서 설치된다. 그 결과차림표를 그림 6-9에 보여준다.

그림 6-9. 표준방조차림표의 배치

"Contents"항목은 지름건 F1을 가지며 방조본문을 현시한다. 본문자체는 HTML로서 작성자가 제공한다. 파일이름은 응용프로그람의 이름에 의존된다. menumain라는 이름의 이 실례에서 영문판 방조나무의 색인파일은 다음과 같다.

/usr/doc/kde/HTML/en/menumain/index.html

"What's This"차림표항목은 현시를 위한 모든 창문부품이 그자체에 대한 설명본문을 현시할수 있도록 선택방식으로 응용프로그람을 절환한다. 이 실례의 24행에서 정적메쏘드 QWhatsThis호출은 기본창문으로 사용된 단추에 설명본문을 삽입한다. 차림표를 선택할 때 유표를 물음표로 변경한다. 물음표는 항목을 선택할 때 본문을 현시하는데 사용된다. 본문은 그림 6-10과 같이 간단한 경계선과 노란색 배경을 가지는 창문에 현시된다.

그림 6-10. "What’s this?"에 대한 응답으로서 현시된 본문

차림표의 밑에 있는 2개의 단추는 About대화칸들이다. 응용프로그람의 기정About대화칸은 OK단추를 가지는 간단한 본문블로크이고 KDE의 현재판에 대한 정보를 가지는 표준 About칸이 있다.

KHelpMenu를 사용하여 기본About칸을 자기의것들중 하나로 바꿀수 있다. 이를 위하여 KHelpMenu클라스를 확장하고 aboutApplication()이라는 처리부를 포함한다. 기정About칸을 펼칠 대신에 이 처리부를 실행하여 자기의 About칸을 만들고 현시할수 있다.

참고: 5장에서 About칸을 만드는 실례를 찾을수 있다.

제3절. 튀여나오기차림표

QPopupMenu객체를 QMenuBar에 련결할 필요는 없다. 차림표를 창문부품의 중심에 펼칠수 있다. 응용프로그람이 할 일은 위치를 지정하고 show()메쏘드를 호출하는것이다. 다음 실례는 그림 6-11과 같이 마우스의 오른쪽단추를 찰칵할 때 튀여나오기차림표를 현시한다.

그림 6-11. 창문부품의 중심에 펼쳐진 차림표

MenuPopup머리부파일

```
1 /* menupopup.h */
2 #ifndef MENUPOPUP_H
3 #define MENUPOPUP_H
4
```

```
5 #include <qpopupmenu.h>
6
7 class MenuPopup: public QWidget
8 {
9     Q_OBJECT
10 public:
11     MenuPopup(QWidget *parent=0,const char *name=0);
12 protected:
13     virtual void mousePressEvent(QMouseEvent *event);
14 private:
15     QPopupMenu *popup;
16 private slots:
17     void slotStub();
18 };
19
20 #endif
```

7행에서 클라스 MenuPopup는 기초클라스로서 QWidget를 사용하고 그것은 창문부품이므로 가상보호메쏘드 mousePressEvent()를 계승한다. 이 함수는 마우스지시자가 창문부품안에 있는데 마우스단추가 눌릴 때마다 호출된다.

MenuPopup

```
1 /* menupopup.cpp */
2 #include <kapplication.h>
3 #include "menupopup.h"
4
5 int main(int argc, char **argv)
6 {
7     KApplication app(argc, argv,  "setxy");
8     MenuPopup menupopup;
9     menupopup.show();
10    app.setMainWidget(&menupopup);
11    return(app.exec());
12 }
13
14 MenuPopup::MenuPopup(QWidget *parent,const char *name)
15    : QWidget(parent,name)
```

```
16 {
17    setMinimumSize(90,40);
18    resize(200,100);
19
20    popup = new QPopupMenu(this);
21    popup->insertItem("Activate",this,SLOT(slotStub()));
22    popup->insertItem("Deactivate",this,SLOT(slotStub()));
23    popup->insertItem("Arrange",this,SLOT(slotStub()));
24    popup->insertSeparator();
25    popup->insertSeparator();
26    popup->insertItem("Logout",this,SLOT(slotStub()));
27 }
28
29 void MenuPopup::mousePressEvent(QMouseEvent *event)
30 {
31    if(event->button() == RightButton) {
32            popup->move(x() + event->x(),y() + event->y());
33            popup->exec();
34    }
35 }
36 void MenuPopup::slotStub() {}
```

14행에서 시작하는 MenuPopup구성자는 QPopupMenu객체를 만들고 클라스선언의 15행에서 정의된 popup에 그 주소를 보관한다. QPopupMenu의 부모창문부품은 보통 차림표를 펼치는 창문부품이다. 21～26행은 차림표에 항목들을 삽입한다.

29행의 가상메쏘드 mousePressedEvent()는 부모QWidget클라스의 메쏘드를 재정의한다. 이 메쏘드는 마우스단추를 찰칵할 때마다 호출된다. 31행은 오른쪽 마우스단추를 선택하였는가를 판단한다. 만일 그렇다면 move()호출은 차림표위치를 지정하고 exec()호출은 그것을 펼친다. 32행에서 move()호출에 공급된 자리표위치는 부모창문부품과 마우스지시자의 x와 y자리표들의 합이다. 결과자리표는 차림표가 직접 마우스아래에 나타나게 하는 화면우의 마우스위치이다.

제4절. 도구띠

KMainWindow는 많은 도구띠들을 관리한다. 기정으로 그것들은 삽입되는 순서로 창문의 꼭대기에 현시된다. 차림표가 있으면 도구띠들은 그 아래에 현시된다. 그리고 도구띠에는 단추그림기호들외에 다른것을 가질수 있다. 다음 실례는 그림 6-12와 같이 단추, 분리선, 복

합칸, 그리고 표식자창문부품을 포함하는 2개 도구띠를 설치한다.

그림 6-12. 여러 항목을 포함하는 한쌍의 도구띠

ToolBarMain머리부파일

```
1 /* main.h */
2 #ifndef TOOLBARMAIN_H
3 #define TOOLBARMAIN_H
4
5 #include <kmainwindow.h>
6 #include <ktoolbar.h>
7
8 class ToolBarMain: public KMainWindow
9 {
10     Q_OBJECT
11 public:
12     ToolBarMain();
13 private slots:
14     void slotExit();
15     void slotStub();
16     void slotFont(int index);
17     bool queryClose();
18 private:
19     void createMainWidget();
20     void createToolBarOne();
21     void createToolBarTwo();
22 };
23
24 #endif
```

그것은 도구띠들을 관리하는 KMainWindow이므로 클라스 ToolBarMain은 8행에서 KMainWindow의 파생클라스로 선언된다. 14～16행에서 정의된 처리부들은 도구띠항목들이 선택될 때 호출을 받아들인다. 17행의 처리부 queryClose()는 체계가 응용프로그람에 창문을 닫으려고 하는가를 묻는데 사용된다.

ToolBarMain

```
1 /* toolbarmain.cpp */
2 #include <kapplication.h>
3 #include <qpushbutton.h>
4 #include <qstrlist.h>
5 #include <qcstring.h>
6 #include "toolbarmain.h"
7
8 int main(int argc, char **argv)
9 {
10    KApplication app(argc, argv,  "toolbarmain");
11    ToolBarMain *toolbarmain = new ToolBarMain();
12    toolbarmain->show();
13    return(app.exec());
14 }
15 ToolBarMain::ToolBarMain() : KMainWindow()
16 {
17    createMainWidget();
18    createToolBarOne();
19    createToolBarTwo();
20 }
21 void ToolBarMain::createMainWidget()
22 {
23    QPushButton *button =
24            new QPushButton("Top Level\nWidget",this);
25    setCentralWidget(button);
26 }
27 void ToolBarMain::createToolBarOne()
28 {
29    QPixmap fpix("flag.png");
30    QPixmap rpix("redo.png");
```

```
31    QPixmap upix("undo.png");
32    QPixmap spix("stop.png");
33    QPixmap epix("exit.png");
34
35    KToolBar *toolbar = toolBar(1);
36    toolbar->insertButton(fpix,5,SIGNAL(clicked()),
37            this,SLOT(slotStub()),TRUE,  "Flag As Used");
38    toolbar->insertButton(rpix,6,SIGNAL(clicked()),
39            this,SLOT(slotStub()),TRUE,  "Redo");
40    toolbar->insertButton(upix,7,SIGNAL(clicked()),
41            this,SLOT(slotStub()),TRUE,  "Undo");
42    toolbar->insertSeparator();
43    toolbar->insertButton(spix,7,SIGNAL(clicked()),
44            this,SLOT(slotStub()),TRUE,  "Stop");
45    toolbar->insertButton(epix,8,SIGNAL(clicked()),
46            this,SLOT(slotExit()),TRUE,  "Exit Program");
47 }
48 void ToolBarMain::createToolBarTwo()
49 {
50    QPixmap fpix("bottom.png");
51    KToolBar *toolbar = toolBar(2);
52    toolbar->insertButton(fpix,10,SIGNAL(clicked()),
53            this,SLOT(slotStub()),TRUE,  "Go To Bottom");
54
55    toolbar->insertLineSeparator();
56
57    QStrList *list = new QStrList();
58    list->insert(0,  "Courier");
59    list->insert(1,  "Times Roman");
60    list->insert(2,  "Arial");
61    toolbar->insertCombo(list,11,FALSE,
62            SIGNAL(activated(int)),
63            this,SLOT(slotFont(int)),
64            TRUE,  "Select Font",110);
65
```

```
66    toolbar->insertSeparator();
67
68    QLabel *label = new QLabel("Any Widget", toolbar);
69    toolbar->insertWidget(12,90,label);
70 }
71 void ToolBarMain::slotExit()
72 {
73    kapp->exit(0);
74 }
75 bool ToolBarMain::queryClose()
76 {
77    return(TRUE);
78 }
79 void ToolBarMain::slotStub() {}
80 void ToolBarMain::slotFont(int index) {}
```

15행에서 시작하는 ToolBarMain구성자는 기본창문용창문부품과 2개의 도구띠를 만들기 위하여 메쏘드들을 호출한다. 21행에 있는 메쏘드 createMainWidget()에서 만들어진 기본창문부품은 누름단추이다.

한개 도구띠성원(우의 도구띠)은 27행에서 메쏘드 createToolBar에 의해 만들어진다. 그림 6-12에 이미 보여준것처럼 이 도구띠의 매개 항목은 자체의 픽스매프를 가지고있다. 픽스매프들은 29∼33행에서 파일들로부터 적재된다. KToolBar는 이 클라스가 KMainWindow로부터 물려받은 메쏘드 toolBar()를 호출하여 만든다. 앞에서 설명한 차림표창문을 만드는 메쏘드와 달리 toolBar()메쏘드는 호출시마다 새 도구띠를 만들어서 돌려준다. 그리고 그것이 돌려주는 도구띠는 이미 현시요소로서 KMainWindow에 삽입되였다.

도구띠의 3번째와 4번째 성원사이에 작은 공간이 있다. 이 공간은 42행에서 insertSeparator()호출에 의해 삽입된다. 더 넓은 공간을 요구한다면 분리선을 더 많이 삽입할 수 있다.

48행에서 메쏘드 createToolBarTwo()에 의해 창조된 둘째 도구띠는 더 복잡한 도구띠단추들을 가지고있다. 표준도구띠단추는 52행에서 insertButton()호출에 의해 추가되고 55행에서 insertLineSeparator()호출에 의해 그 오른쪽에 수직분리선이 삽입된다.

문자렬배렬은 58∼60행에서 insert()호출에 의해 QStrList객체에 삽입되고 그 배렬은 61행에서 insertCombo()을 호출함으로써 도구띠의 복합칸을 설치하는데 사용된다. 내부적으로 KToolBar는 목록을 관리하기 위하여 표준QComboBox를 창조하므로 insertCombo()에 넘긴 대부분의 정보는 QComboBox에 넘겨진다. 58∼60행의 insert()메쏘드들은 복합칸본문을 정의하고 매개 본문에 ID번호를 할당한다. 61행의 insertCombo()호출은 목록을 사용되여 ID번호 11

을 가지는 QComboBox를 만들며 사용자가 써넣기할수 없다는것을 지정한다. 신호는 activated()이고 처리부는 slotFont()이며 그것들은 둘다 선택항목의 ID번호를 처리부메쏘드에 넘기기 위한 int인수를 가진다. TRUE인수는 QComboBox가 허용된다는것을 지정한다. 문자렬 "Select Font"는 도구암시의 본문을 지정하며 수값110은 화소너비를 지정한다.

알아두기: 이 실례의 복합칸은 ID번호를 가지고 처리부를 호출하지만 대신에 문자렬을 사용할수 있다. 이를 위하여 신호로서 activated(String &), 처리부로서 slotFont(String &)를 사용한다.

도구띠에 어떠한 창문부품이든지 설치할수 있다. 실례로 insertCombo()를 호출하는것외에 QComboBox를 더 많이 조종하려고 한다면 자체의 복합칸을 만들고 insertWidget()를 호출하여 그것을 설치할수 있다. 68행과 69행의 코드는 표식자를 만들고 설치한다. 표식자는 다른 항목들이 수행하는 방식으로 마우스에 응답하지 않는다. 창문부품을 설치하고 있을 때 도구띠는 응답을 얻을 필요가 있는 모든 신호와 처리부들을 설정한다고 가정한다.

제5절. 상태띠

KMainWindow의 요소로 포함할수 있는 KStatusBar창문부품이 있다. 그것은 보통 창문바닥에 현시되며 응용프로그람이 본문행을 현시하고 계속 갱신하는데 사용될수 있다. 그림 6-13에 보여주는 다음 실례는 상태띠를 사용하여 한쌍의 단추들에 의해 증가감소되는 내부계수기의 현재값을 현시한다.
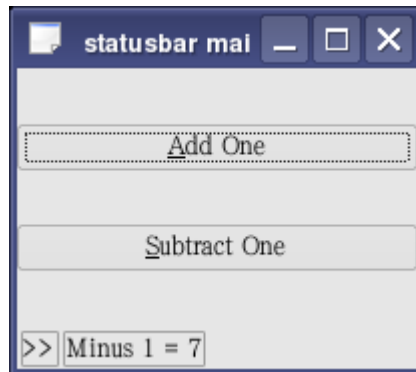
그림 6-13. 값을 추적하는 상태띠

StatusBarMain머리부파일

```
1 /* statusbarmain.h */
2 #ifndef STATUSBARMAIN_H
3 #define STATUSBARMAIN_H
4
5 #include <kmainwindow.h>
6 #include <kstatusbar.h>
7
```

```
8 class StatusBarMain: public KMainWindow
9 {
10    Q_OBJECT
11 public:
12    StatusBarMain();
13 private:
14    int counter;
15    KStatusBar *status;
16 private slots:
17    bool queryClose();
18    void slotAddOne();
19    void slotSubtractOne();
20 private:
21    void createMainWidget();
22    void createStatusBar();
23 };
24
25 #endif
```

8행에서 선언이 시작되는 클라스 StatusBarMain은 KMainWindow클라스를 계승하며 이것은 KStatusBar를 계승한다는것을 의미한다. 추적하는 값은 14행에서 counter로 정의되고 호출하기 쉽게 KStatusBar지적자는 15행에서 정의된 status에 보관된다.

StatusBarMain

```
1 /* statusbarmain.cpp */
2 #include <kapplication.h>
3 #include <qpushbutton.h>
4 #include <kcontainer.h>
5 #include "statusbarmain.h"
6
7 int main(int argc, char **argv)
8 {
9     KApplication app(argc, argv,  "statusbarmain");
10    StatusBarMain *statusbarmain = new StatusBarMain();
11    statusbarmain->show();
12    return(app.exec());
13 }
```

```
14 StatusBarMain::StatusBarMain() : KMainWindow()
15 {
16     counter = 0;
17     createMainWidget();
18     createStatusBar();
19 }
20 void StatusBarMain::createMainWidget()
21 {
22     KContainerLayout *layout =
23             new KContainerLayout(this,  "layout");
24     layout->setOrientation(KContainerLayout::Vertical);
25
26     QPushButton *button;
27     button = new QPushButton("Add One",this);
28     connect(button,SIGNAL(clicked()),
29             this,SLOT(slotAddOne()));
30     layout->packStart(button);
31     button = new QPushButton("Subtract One",this);
32     connect(button,SIGNAL(clicked()),
33             this,SLOT(slotSubtractOne()));
34     layout->packStart(button);
35
36     layout->sizeToFit();
37     setCentralWidget(layout);
38 }
39 void StatusBarMain::createStatusBar()
40 {
41     status = statusBar();
42     status->insertItem(QString(">>"),1);
43     status->insertItem(QString("Add or Subtract"),2);
44 }
45 void StatusBarMain::slotAddOne()
46 {
47     status->changeItem(
48             QString("Plus 1 = %1").arg(++counter),2);
```

```
49 }
50 void StatusBarMain::slotSubtractOne()
51 {
52     status->changeItem(
53             QString("Minus 1 = %1").arg(--counter),2);
54 }
55 bool StatusBarMain::queryClose()
56 {
57     return(TRUE);
58 }
```

프로그람의 기본함수는 StatusBarMain객체를 만든다. 그것은 KMainWindow를 계승하므로 자동적으로 응용프로그람의 제일웃준위창문으로 된다. 14행에서 시작하는 StatusBarMain구성자는 내부계수기값을 0으로 초기화한 다음 기본창문부품을 만들고 상태띠의 초기본문을 설치한다.

20행에서 시작하는 메쏘드 createMainWidget()는 그림 6-13에 이미 보여준것처럼 KContainerLayout창문부품을 사용하여 2개 단추를 보유한다. 하나의 단추는 처리부메쏘드 slotAddOne()에 련결되고 다른 단추는 slotSubtractOne()에 련결된다.

상태띠자체는 39행의 메쏘드 createStatusBar()에서 초기화된다. statusBar()호출은 상태띠의 실례를 만들고 그 지적자를 돌려준다. KMainWindow에 하나의 KStatusBar만 있을수 있으므로 statusBar()에 대한 다음 호출은 같은 상태띠객체의 주소를 되돌려준다. 사실상 이 클라스내로부터 상태띠를 호출하려고만 한다면 항상 지적자를 얻을수 있으므로 자체로 지적자를 보관할 필요는 없다.

42행과 43행은 상태띠의 insertItem()을 호출하여 현시문자렬을 삽입한다. 문자렬은 두 부분(더 있을수 있다)에 삽입되고 매개에 ID번호가 할당된다. 문자렬들은 각각 추가되는 순서로 현시되고 그것들을 변경하려고 할 때 ID번호가 요구된다. 이리하여 그것들을 모두 바꾸지 않고 문자렬부분을 변경한다. 실례로 45행의 처리부메쏘드 slotAddOne()는 "Add"단추가 선택될 때마다 호출되여 계수기를 증가시키며 ID번호 2를 가지는 상태띠본문은 changeItem() 호출에 의해 교체된다. ID번호 1을 가지는 본문은 바뀌지 않는다. 마찬가지로 "Subtract"단추는 50행에서 slotSubtractOne()을 실행하여 계수기를 감소시키며 ID번호2의 본문만 변경한다.

알아두기: QString클라스의 arg()메쏘드가 문자렬들을 여러가지 자료형들로 형식화하는데 사용된다. 모든 자료형과 형식선택의 실례는 편의클라스들을 설명하는 16장에서 설명한다.

요구된다면 많은 본문구획(또는 본문항목)들로 현시문자렬을 분리하고 매개 문자렬과 개별적으로 작업할수 있다. 또한 KStatusBar메쏘드clear()를 호출하여 모든 ID번호들로부터 본문을 삭제할수 있다.

요 약

KMainWindow클라스는 응용프로그람의 기능호출을 사용자에게 제공하는데 쓰이는코드를 포함하는 특수한 제일웃준위창문이며 사용자가 응용프로그람의 현재상태를 계속 알수 있게 한다.

- 차림표띠는 제일웃준위창문의 꼭대기나 바닥에 현시될수 있다. 사용자의 요구에 따라 그것은 제일웃준위창문에서 떼내여 화면에 독립적인 존재로서 표시할수 있다.
- 어떠한 창문이든지 마우스나 건반의 조종하에 펼쳐지는 차림표를 가질수 있다. 이 차림표로부터의 선택 또는 다른 위치에서 마우스찰칵은 펼쳐진 차림표를 지워버린다.
- 많은 도구띠들은 제일웃준위창문의 4개 측면의 어디에나 개별적으로 위치할수 있으며 또는 개개를 분리하여 화면에서 독립적인 항목으로서 현시할수 있다.
- 상태띠는 항상 갱신된 본문을 가지며 제일웃준위창문의 바닥에 현시되게 만들어진다.

7장은 임의의 순간에 오직 하나만 선택할수 있는 라지오단추그룹과 같이 서로 련관된 집합을 만들고 현시하는데 쓰이는 창문부품들과 목록으로부터 하나이상의 항목들을 사용자가 선택하는데 쓰이는 복합칸을 설명한다.

제7장. 창문부품그룹

학습내용

- 하나의 처리부에 의해 동작하도록 일련의 단추들을 조직하는 방법
- 라지오단추들을 련동시키는 방법
- 검사단추들을 련결하는 방법
- 틀을 창문부품에 구축하는 방법
- 자기 창문부품들을 틀로 장식하는 방법
- 분리띠들을 이동하여 여러 창문부품들의 크기를 변경하는 방법

이 장은 창문이 자주 배치될 때 흔히 제기되는 문제들을 해결하는데 사용할수 있는 몇가지 창문부품과 용기들을 시험한다. 실례로 여러개의 단추들을 가진 하나의 창문부품을 만들고 이 단추들을 모두 같은 처리부에 련결할수 있다. 라지오단추들중 하나의 선택은 그룹안의 다른 라지오단추의 선택을 해제하므로 그것들을 서로 련동시켜야 한다. 가끔 창문부품그룹은 그것들이 수행하는 기능에 따라 련결되며 그것들을 둘러싸는 틀을 그리여 사용자에게 보여주는 방법이 있다.

제1절. KButtonBox

대화칸창문의 아래에 단추들을 배치하는것은 아주 례사로운 일이다. 클라스 KButtonBox는 단추들의 위치를 지정하는 일을 간단화하는 용기창문부품이다. 다음 실례는 KButtonBox를 사용하여 수평으로 배렬된 3개 단추를 관리한다.

```
1 /* hbuttonbox.cpp */
2 #include <kapplication.h>
3 #include <kbuttonbox.h>
4
5 int main(int argc, char **argv)
6 {
7     QPushButton *button1;
8     QPushButton *button2;
9     QPushButton *button3;
10    KApplication app(argc, argv,  "vbuttonbox");
11
12    KButtonBox *box =
13            new KButtonBox(0,KButtonBox::HORIZONTAL,25,15);
14    button1 = box->addButton("First Button");
15    button2 = box->addButton("Second Button");
```

```
16    button3 = box->addButton("Third Button");
17    box->layout();
18    box->show();
19    box->resize(10,10);
20
21    app.setMainWidget(box);
22    return(app.exec());
23 }
```

단추칸은 12행과 13행에서 만들어진다. 첫 인수는 보통 부모창문부품의 주소이지만 실례에서는 그것을 제일웃준위창문으로 사용하므로 부모가 없다. 둘째 인수는 단추들의 방향 즉 HORIZONTAL 또는 VERTICAL을 지정한다. 마지막 2개 인수는 단추들의 주위에 삽입하는 최소공간을 지정한다. 첫째 인수는 매개 단추와 KButtonBox의 경계사이의 최소화소거리를 지정한다. 둘째 인수는 단추들사이의 화소거리를 지정한다. 결과는 그림 7-1에 보여준 단추들의 행이다.
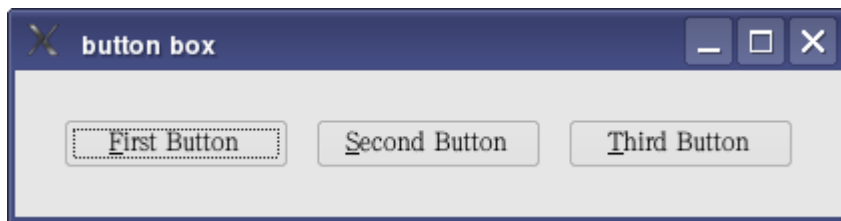
그림 7-1. KButtonBox에 포함된 3개의 단추

14～16행에서 addButton()호출은 단추들을 만든다. 즉 KButtonBox는 매개 단추를 만들고 그것의 지적자를 돌려준다. 실례를 간단하게 만들기 위하여, 단추들을 처리부에 련결하지 않는다. 17행의 layout()호출은 그밖에 아무것도 추가하려고 하지 않다는것과 나아가서 자체의 환경을 구성한다는것을 KButtonBox에게 알리기 위하여 필요하다. 18행과 19행은 임의의 다른 창문부품에서나 동일하며 KButtonBox는 그 자체(그리고 그것의 모든 내용)를 현시하게 하고 그 최소높이와 최소너비로 크기가 설정된다.

그림 7-1의 단추들은 모두 크기가 같다. 포함하는 본문의 길이에 따라 단추너비를 변경하려고 한다면 addButton()메쏘드에 둘째 인수로서 true를 지정할수 있다. 실례로 다음 코드는 단추들이 담고있는 본문에 따라서 너비가 달라지도록 한다.

```
button1 = box->addButton("First Button",TRUE);
button2 = box->addButton("Second Button",TRUE);
button3 = box->addButton("Third Button",TRUE);
```

둘째인수를 FALSE(기정)로 하면 KButtonBox는 가장 넓은 단추의 크기를 결정하고 다른 것들의 크기는 그것과 일치하게 조절한다. 드문히 KButtonBox의 너비를 그것을 담고있는 대화칸의 너비와 정확히 일치시킴으로써 그것이 자체를 변경시키는 방법과 위치를 지정할수

있다. 그림 7-2의 꼭대기에 있는 배치는 단추들이 모두 왼쪽에 놓여있는 기정변화를 보여준다. 그림의 바닥에 있는 배치는 가변점을 지정한 경우를 보여준다.

그림 7-2. 변경이 정의된 경우와 안된 경우의 KButtonBox

그러기 위하여 다음과 같이 두 단추들사이에 가변점을 삽입한다.

```
button1 = box->addButton("First Button");
button2 = box->addButton("Second Button");
box->addStretch(1)
button3 = box->addButton("Third Button");
```

신축은 3장에서 론의된 용기들에서처럼 수행된다. 즉 필요한만큼 가변점들을 추가할수 있고 매개는 다른것들에 비례하여 변경되며 비례는 addStretch()의 인수값에 의해 결정된다.

조금 변경하면 이전 실례는 수직방향으로 변환될수 있다. 즉 구성자의 둘째 인수를 VERTICAL로 변경한다.

```
KButtonBox *box = new KButtonBox(0,KButtonBox::VERTICAL,25,15);
```

결과창문을 그림 7-3에 보여준다. 변두리로부터 그리고 단추들을 서로 분리하는 공간은 같은 방법으로 지정된다. 창문이 수직으로 변화될 때 KButtonBox는 바닥 또는 변화를 지정하는 위치에 공간을 삽입한다.

그림 7-3. 수직방향을 가지는 KButtonBox

제2절. 하나의 처리부를 가지는 단추그룹

QButtonGroup객체는 수평 또는 수직으로 단추그룹을 배렬하는데 사용될수 있다. 그룹에 추가된 매개 단추에는 ID번호가 할당되고 필요하다면 모든 단추들에 대하여 하나의 처리부 메쏘드를 사용할수 있다. 직접 QButtonGroup을 창조할수 있지만 단추들을 수평으로 배렬하는가 수직으로 배렬하는가에 따라 QHButtonGroup이나 QVButtonGroup을 사용하는것이 더 간단하다. 또한 QButtonGroup은 QFrame이므로 QFrame메쏘드호출을 사용하여 그룹의 모양을 바꿀수 있다.

다음 실례는 수평QButtonGroup내에 4개 단추를 가진다. 그림 7-4에 보여준것처럼 단추들의 행아래에는 매개 단추를 누를 때마다 본문이 갱신되는 표식자가 있다.
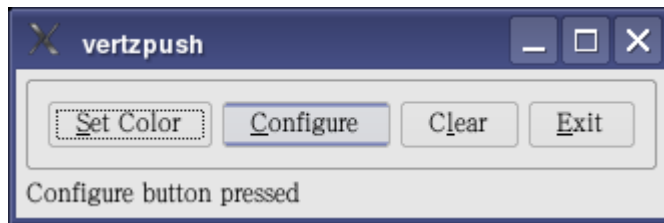
그림 7-4. 단추들을 수평으로 배렬하는 QHButtonGroup창문부품

HorizPush머리부파일

```
1 /* horizpush.h */
2 #ifndef HORIZPUSH_H
3 #define HORIZPUSH_H
4
5 #include <qwidget.h>
6 #include <qlabel.h>
7
8 class HorizPush: public QWidget
9 {
10    Q_OBJECT
11 public:
12    HorizPush(QWidget *parent=0,const char *name=0);
13 private:
14    QLabel *label;
15    enum ButtonChoice { SetColor, Configure, Clear, Exit };
16 private slots:
17    void slotButton(int ID);
18 };
```

```
19
20 #endif
```

14행에서 선언된 표식자는 창문의 바닥에 본문을 현시한다. 15행의 렬거선언은 매개 단추에 련결된 ID번호로서 사용되므로 처리부메쏘드는 어느 단추를 눌렀는가 결정할수 있다.

HorizPush

```
1 /* horizpush.cpp */
2 #include <kapplication.h>
3 #include <qlayout.h>
4 #include <qhbuttongroup.h>
5 #include <qpushbutton.h>
6 #include "horizpush.h"
7
8 int main(int argc, char **argv)
9 {
10    KApplication app(argc, argv,  "horizpush");
11    HorizPush *horizpush = new HorizPush();
12    horizpush->show();
13    app.setMainWidget(horizpush);
14    return(app.exec());
15 }
16
17 HorizPush::HorizPush(QWidget *parent,const char *name)
18    : QWidget(parent,name)
19 {
20    QPushButton *button;
21    QVBoxLayout *layout = new QVBoxLayout(this,5);
22
23    QHButtonGroup *group = new QHButtonGroup(this,  "hg1");
24    button = new QPushButton("Set Color",group);
25    group->insert(button,SetColor);
26    button = new QPushButton("Configure",group);
27    group->insert(button,Configure);
28    button = new QPushButton("Clear",group);
29    group->insert(button,Clear);
30    button = new QPushButton("Exit",group);
```

```
31    group->insert(button,Exit);
32    connect(group,SIGNAL(clicked(int)),
33            this,SLOT(slotButton(int)));
34    layout->addWidget(group);
35
36    label = new QLabel("  ",this);
37    layout->addWidget(label);
38
39    resize(10,10);
40    layout->activate();
41 }
42 void HorizPush::slotButton(int ID)
43 {
44    switch(ID) {
45            case SetColor:
46                    label->setText("Set Color button pressed");
47                    break;
48            case Configure:
49                    label->setText("Configure button pressed");
50                    break;
51            case Clear:
52                    label->setText(" ");
53                    break;
54            case Exit:
55                    kapp->exit(0);
56    }
57 }
```

17행에서 시작하는 구성자는 창문배치를 만든다. 기본배치관리기는 21행에서 만들어지는 수직칸이고 2개 창문부품 즉 우에 QHButtonGroup, 아래에 QLabel을 포함한다.

QHButtonGroup은 23행에서 HorizPush객체를 부모창문부품으로 하여 만들어지므로 QVBoxLayout객체가 용기로서 작용한다 할지라도 창문부품이 아니다. HorizPush객체는 QWidget를 계승하므로 다른 창문부품의 부모로서 쓰일수 있다.

24～31행은 4개 누름단추를 만들어서 QHButtonGroup에 삽입한다. QHButtonGroup도 창문부품이므로 누름단추창문부품들의 부모로서 쓰일수 있다. insert()호출은 매개 단추를 그룹에 삽입할 때 ID번호를 할당한다. ID번호를 지정하지 않으면 첫째 단추는 기정으로 0, 둘째

는 1, …으로 된다. 그러나 ID번호는 단추들을 식별하는 유일한 방법이므로 그것들을 자기가 지정하는것이 좋다. 이 실례에서 값들을 렬거형값으로 정의하여 단추들을 추가삭제하는 일을 편리하게 한다.

32행에서 connect()호출은 QHButtonGroup의 checked()신호를 국부처리부 slotButton()에 련결한다. 34행에서 QHButtonGroup은 수직칸의 꼭대기에 삽입된다. 36행과 37행에서 수직칸의 바닥에 표식자를 만들어서 보관한다.

내부적으로 QHButtonGroup은 매개 단추로부터 clicked()신호를 받아들이는 처리부를 가지고 있으며 자기의 clicked()신호를 발생하여 단추에 할당된 ID번호를 전한다. 42행에서 slotButton()처리부는 신호를 받아들이고 ID번호를 리용하여 어떤 작용이 취해졌는가를 판단한다. ID가 SetColor 또는 Configure이면 본문이 설정된다. ID가 Clear이면 본문은 지워진다. Exit값은 프로그람을 완료한다.

QHButtonGroup창문부품은 단추들을 수평으로 현시하는데 쓰이고 QVButtonGroup은 수직으로 현시하는데 쓰인다. 필요한 과정은 수평그룹을 만드는 과정과 꼭같다. 그림 7-5에 보여주는 창문을 현시하도록 이전 실례를 변경하기 위하여 수평단추그룹대신에 수직단추그룹을 포함하여 4행을 변경한다.

```
#include <qhbuttongroup.h>
```

그다음 23행을 변경하여 수평단추그룹대신에 수직단추그룹을 다음과 같이 만든다.

```
QVButtonGroup *group = new QVButtonGroup ( this, "vg1") ;
```
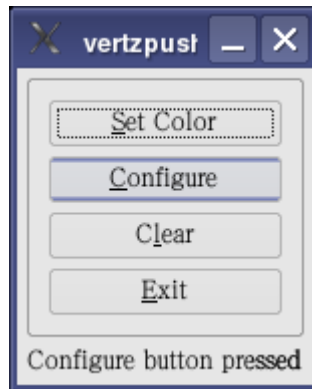
그림 7-5. 수직으로 단추들을 배렬하는 QVButtonGroup창문부품

제3절. 라지오단추그룹만들기

QVButtonGroup은 라지오단추들의 수직렬사이의 관계를 조종하는데 쓰이고 QHButtonGroup은 수평행을 조종하는데 쓰인다. QRadioButton을 QButtonGroup에 삽입할 때마다 한개 단추를 선택할 때와 같은 방법으로 다른 단추들과 련결된다. 그림 7-6에 보여준 창문은 다음 실례에 의해 생성된다.
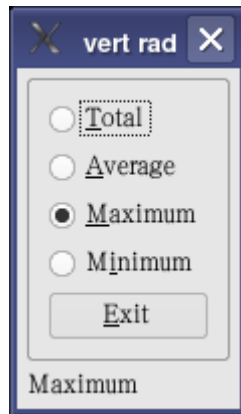
그림 7-6. 라지오단추들의 모임을 조종하는 QVButtonGroup창문부품

VertRadio머리부파일

```
1 /* vertradio.h */
2 #ifndef VERTRADIO_H
3 #define VERTRADIO_H
4
5 #include <qwidget.h>
6 #include <qlabel.h>
7
8 class VertRadio: public QWidget
9 {
10     Q_OBJECT
11 public:
12     VertRadio(QWidget *parent=0,const char *name=0);
13 private:
14     QLabel *label;
15     enum ButtonChoice { Total, Average,
16             Maximum, Minimum, Exit };
17 private slots:
18     void slotButton(int ID);
19 };
20
21 #endif
```

매개 단추는 유일식별번호를 가지며 15행에서 선언된 렬거 ButtonChoice값은 자기 프로그람이 이름에 의해 매개 번호를 참고할수 있게 한다.

VertRadio

```
1 /* vertradio.cpp */
2 #include <kapplication.h>
3 #include <qlayout.h>
4 #include <qvbuttongroup.h>
5 #include <qradiobutton.h>
6 #include <qpushbutton.h>
7 #include "vertradio.h"
8
9 int main(int argc, char **argv)
10 {
11     KApplication app(argc, argv,  "vertradio");
12     VertRadio *vertradio = new VertRadio();
13     vertradio->show();
14     app.setMainWidget(vertradio);
15     return(app.exec());
16 }
17
18 VertRadio::VertRadio(QWidget *parent,const char *name)
19     : QWidget(parent,name)
20 {
21     QRadioButton *button;
22     QVBoxLayout *layout = new QVBoxLayout(this,5);
23
24     QVButtonGroup *group = new QVButtonGroup(this,  "vg1");
25     button = new QRadioButton("Total",group);
26     group->insert(button,Total);
27     button = new QRadioButton("Average",group);
28     group->insert(button,Average);
29     button = new QRadioButton("Maximum",group);
30     group->insert(button,Maximum);
31     button = new QRadioButton("Minimum",group);
32     group->insert(button,Minimum);
33     QPushButton *pButton = new QPushButton("Exit",group);
34     group->insert(pButton,Exit);
35     connect(group,SIGNAL(clicked(int)),
```

```
36          this,SLOT(slotButton(int)));
37    layout->addWidget(group);
38
39    label = new QLabel("  " ,this);
40    layout->addWidget(label);
41
42    resize(10,10);
43    layout->activate();
44 }
45 void VertRadio::slotButton(int ID)
46 {
47    switch(ID) {
48          case Total:
49                label->setText("Total");
50                break;
51          case Average:
52                label->setText("Average");
53                break;
54          case Maximum:
55                label->setText("Maximum");
56                break;
57          case Minimum:
58                label->setText("Minimum");
59                break;
60          case Exit:
61                kapp->exit(0);
62    }
63 }
```

이 실례는 VertRadio객체를 만들어서 제일웃준위창문의 창문부품으로 사용한다. 18행에서 시작하는 구성자는 QVBoxLayout를 사용하여 아래에 표식자를 가지는 단추들의 목록을 포함한다. 표식자는 어느 단추가 능동으로 되는가를 알리는데 사용된다.

24～32행은 4개 라지오단추를 만들어서 QVButtonGroup에 삽입한다. 표준QPushButton은 33행에서 만들어지고 34행에서 같은 QVButtonGroup창문부품에 설치된다. 단추들을 만들 때 QVButtonGroup를 부모창문부품으로 사용한다. QButtonGroup의 모든 라지오단추들은 자동적으로 련동되므로 한번에 하나씩 선택된다. QVButtonGroup는 오직 라지오단추들만 련동시키

고 다른 종류의 단추들은 독자적인 실체로 남아있으므로 그룹에 여러가지 형의 단추들을 섞을수 있다.

45행에서 정의된 처리부메쏘드 slotButton()은 모든 단추들에 대하여 호출되며 그것들의 형에는 관계없다. 단추의 ID값을 검사한 후 처리부메쏘드는 현재 어느 라지오단추가 선택되였는가를 가리키도록 표식자본문을 설정한다. 비라지오단추는 프로그람을 완료하는데 사용될수 있다.

라지오단추들을 수직으로 배렬하는것이 관례이지만 만약 그것들을 수평으로 배렬해야 한다면 간단히 수행할수 있다. 이전 실례를 수평방향으로 바꾼 결과를 그림 7-7에 보여준다. 그러기 위하여 5행을 다음과 같이 변환한다.

```
#include <qhbuttongroup.h>
```

그리고 24행을 수평그룹칸의 창조로 변환한다.

```
QHButtonGroup *group = new QHButtonGroup(this, "hg1") ;
```
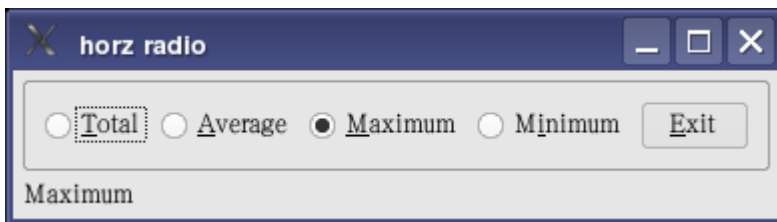
그림 7-7. 수평으로 배렬된 라지오단추그룹

제4절. 검사단추그룹만들기

QCheckBox는 off와 on상태를 절환할수 있는 단추이다. 상태는 QCheckBox자체에 보관된다. 검사단추는 가끔 절환단추라고 부른다. 다음 실례는 그림 7-8에 보여주는 검사단추들의 그룹을 만든다. 검사표식자는 검사단추가 on상태로 되여야만 나타난다.
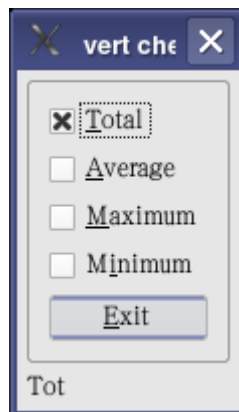
그림 7-8. 수직으로 배렬된 QCheckBox단추그룹

VertCheck머리부파일

```
1 /* vertcheck.h */
```

```
2 #ifndef VERTCHECK_H
3 #define VERTCHECK_H
4
5 #include <qwidget.h>
6 #include <qlabel.h>
7 #include <qvbuttongroup.h>
8
9 class VertCheck: public QWidget
10 {
11     Q_OBJECT
12 public:
13     VertCheck(QWidget *parent=0,const char *name=0);
14 private:
15     QVButtonGroup *group;
16     QLabel *label;
17     enum ButtonChoice { Total, Average,
18             Maximum, Minimum, Exit };
19     bool totalFlag;
20     bool averageFlag;
21     bool minimumFlag;
22     bool maximumFlag;
23 private slots:
24     void slotButton(int ID);
25 };
26
27 #endif
```

QVButtonGroup은 15행에서 클라스의 자료성원으로 포함된다. 그것은 단추정보를 받아들이는 처리부가 검사단추상태를 group으로부터 얻는데 필요한 단추ID번호를 공급하기때문이다. 현재 QCheckButton의 상태설정값은 19～22행에서 선언된 Boolean변수들에 보관된다.
VertCheck

```
1 /* vertcheck.cpp */
2 #include <kapplication.h>
3 #include <qlayout.h>
4 #include <qcheckbox.h>
5 #include <qpushbutton.h>
```

```
6 #include "vertcheck.h"
7
8 int main(int argc, char **argv)
9 {
10    KApplication app(argc, argv,  "vertcheck");
11    VertCheck *vertcheck = new VertCheck();
12    vertcheck->show();
13    app.setMainWidget(vertcheck);
14    return(app.exec());
15 }
16
17 VertCheck::VertCheck(QWidget *parent,const char *name)
18    : QWidget(parent,name)
19 {
20    QCheckBox *button;
21    QVBoxLayout *layout = new QVBoxLayout(this,5);
22
23    group = new QVButtonGroup(this,  "vg1");
24    button = new QCheckBox("Total",group);
25    group->insert(button,Total);
26    button = new QCheckBox("Average",group);
27    group->insert(button,Average);
28    button = new QCheckBox("Maximum",group);
29    group->insert(button,Maximum);
30    button = new QCheckBox("Minimum",group);
31    group->insert(button,Minimum);
32    QPushButton *pButton = new QPushButton("Exit",group);
33    group->insert(pButton,Exit);
34    connect(group,SIGNAL(clicked(int)),
35            this,SLOT(slotButton(int)));
36    layout->addWidget(group);
37
38    label = new QLabel("  ",this);
39    layout->addWidget(label);
40
```

```
41    totalFlag = FALSE;
42    averageFlag = FALSE;
43    minimumFlag = FALSE;
44    maximumFlag = FALSE;
45
46    resize(10,10);
47    layout->activate();
48 }
49 void VertCheck::slotButton(int ID)
50 {
51    QButton *button = group->find(ID);
52    switch(ID) {
53            case Total:
54                    totalFlag = ((QCheckBox *)button)->isChecked();
55                    break;
56            case Average:
57                    averageFlag = ((QCheckBox *)button)->isChecked();
58                    break;
59            case Maximum:
60                    maximumFlag = ((QCheckBox *)button)->isChecked();
61                    break;
62            case Minimum:
63                    minimumFlag = ((QCheckBox *)button)->isChecked();
64                    break;
65            case Exit:
66                    kapp->exit(0);
67    }
68    QString string;
69    if(totalFlag)
70            string += QString("Tot ");
71    if(averageFlag)
72            string += QString("Avg ");
73    if(maximumFlag)
74            string += QString("Max ");
75    if(minimumFlag)
```

```
76            string += QString("Min ");
77    label->setText(string);
78 }
```

17행에서 시작하는 VertCheck구성자는 수직칸용기를 만들고 그 꼭대기창문부품으로서 QVButtonGroup, 바닥창문부품으로서 QLabel을 설치한다. QVButtonGroup의 주소는 클라스에서 group에 보관된다. addWidget()호출은 QVButtonGroup을 QVBoxLayout배치관리기의 꼭대기창문부품으로 만든다.

24～31행은 4개 QCheckBox객체를 창조하여 QVButtonGroup에 삽입한다. 32～35행에서 5번째로 표준누름단추를 만들어 group에 삽입한다. 38행과 39행에서 창문바닥에 본문을 현시하는데 사용할 표식자를 만들고 layout에 설치한다.

QCheckBox의 기정조건은 off이고 이것은 FALSE를 표시하므로 41～44행은 4개 내부기발을 검사칸과 같은 값으로 설정하는데 사용된다. 하나이상의 검사칸을 초기에 on으로 미리 설치하려고 한다면 다음과 같이 할수 있다.

```
button->setChecked(TRUE);
```

그다음 그것의 대응하는 Boolean값을 TRUE로 설정한다. 다음 절에서 설명하는것처럼 둘이상의 상태가 검사칸에서 가능하다.

49행의 처리부메쏘드 slotButton()은 검사단추들중 하나가 절환될 때마다 호출된다. 활성화된 단추의 ID값은 인수로 제공되며 이 메쏘드는 검사단추의 내부상태를 결정하는데 필요하므로 51행에서 find()호출은 검사칸자체의 주소를 얻는데 사용된다. 52행의 switch문은 어느 단추를 선택하였는가 결정하는데 사용된다. 단추가 검사칸이라면 isChecked()호출은 단추가 on이면 TRUE를, off이면 FALSE를 돌려준다. 국부Boolean변수에 검사칸상태를 보관함으로써 프로그람이 모든 단추들의 상태를 빨리 호출하게 한다.

ID값이 그것이 Exit단추라는것을 가리킨다면 65행의 case문이 실행되여 exit()호출이 응용프로그람을 중지하게 한다.

68～77행은 어느 절환단추가 현재 on인가를 지정하는 문자렬을 만들고 문자렬을 초기에 그림 7-8에 보여준 창문바닥에 현시된 표식자본문으로 설정한다.

대체로 절환단추그룹은 수직으로 배렬되지만 수평으로 배렬하려는 경우가 있다. 이전 실례는 7행의 vertcheck.h를 다음과 같이 변경하여 그림 7-9과 같이 수평으로 구성할수 있다.

```
#include <qhbuttongroup.h>
```

또한 15행의 vertcheck.h를 다음과 같이 변경한다.

```
QHButtonGroup *group;
```

또한 23행의 vertcheck.cpp를 다음과 같이 변경한다.

```
group = QHButtonGroup(this,  "hg1");
```
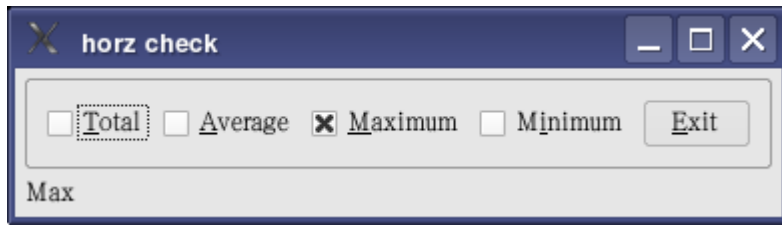
그림 7-9. 수평으로 배렬된 QCheckBox단추그룹

제5절. 틀로서 창문부품

만약 창문부품집합을 틀이나 칸에 포함하여 창문부품들을 련결하고 하나의 단위로서 동작하게 하거나 같은 창문안의 다른 창문부품들과 분리해야 한다면 QFrame창문부품은 그것들을 한개 칸에 포함시키는데 사용될수 있다. QFrame의 바깥에 어떤 창문부품도 남지 않을 때에도 틀의 장식은 창문의 전체 모양을 개량할수 있다.

계승나무에서 QFrame의 직계기초클라스는 QWidget이다. 이것은 자기가 만드는 창문부품의 기초클라스를 QWidget가 아니라 QFrame로 하여 가능하다는것을 의미한다. 그리고 현존 창문부품들의 대부분은 이러한 방법으로 이미 만드느것이다. 실례로 QLabel창문부품은 QFrame를 기초클라스들로 사용하지만 기정으로 장식을 허용하지 않고 표식자에 틀을 추가하는것은 간단히 형과 크기를 지정하는 문제이다. 다음 실례는 그림 7-10과 같이 틀을 허용하는 4개의 표식자를 보여주는 창문을 현시한다.
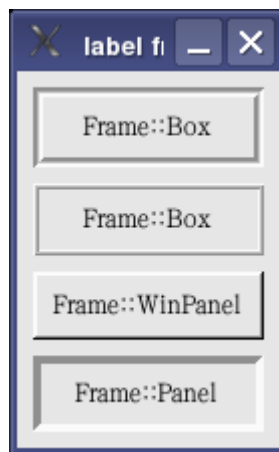
그림 7-10. 틀을 가진 4개의 표식자들

LabelFrame

```
1 /* labelframe.cpp */
2 #include <qlayout.h>
3 #include <qframe.h>
4 #include <kapplication.h>
5 #include <qlabel.h>
```

```
6 #include “labelframe.h”
7
8 int main(int argc, char **argv)
9 {
10    KApplication app(argc, argv, "labelframe");
11    LabelFrame *labelframe = new LabelFrame();
12    labelframe->show();
13    app.setMainWidget(labelframe);
14    return(app.exec());
15 }
16
17 LabelFrame::LabelFrame(QWidget *parent,const char *name)
18    : QWidget(parent,name)
19 {
20    QLabel *lab;
21    QVBoxLayout *layout = new QVBoxLayout(this,8);
22
23    lab = new QLabel("QFrame::Box",this);
24    lab->setFrameStyle(QFrame::Box | QFrame::Sunken);
25    lab->setLineWidth(2);
26    lab->setMidLineWidth(1);
27    lab->setAlignment(AlignVCenter | AlignHCenter);
28    lab->setMargin(8);
29    layout->addWidget(lab);
30
31    lab = new QLabel("QFrame::Box",this);
32    lab->setFrameStyle(QFrame::Box | QFrame::Raised);
33    lab->setLineWidth(1);
34    lab->setMidLineWidth(1);
35    lab->setAlignment(AlignVCenter | AlignHCenter);
36    lab->setMargin(8);
37    layout->addWidget(lab);
38
39    lab = new QLabel("QFrame::WinPanel",this);
40    lab->setFrameStyle(QFrame::WinPanel | QFrame::Raised);
```

```
41    lab->setAlignment(AlignVCenter | AlignHCenter);
42    lab->setMargin(8);
43    layout->addWidget(lab);
44
45    lab = new QLabel("QFrame::Panel",this);
46    lab->setFrameStyle(QFrame::Panel | QFrame::Sunken);
47    lab->setAlignment(AlignVCenter | AlignHCenter);
48    lab->setLineWidth(4);
49    lab->setMargin(8);
50    layout->addWidget(lab);
51
52    resize(10,10);
53    layout->activate();
54 }
```

17행에서 시작하는 구성자는 수직칸배치관리기를 만들고 4개의 표식자를 거기에 포함한다.

23～29행은 Sunken그림자와 Box형태를 결합하여 표식자를 만든다. 결과패턴은 마치도직4각형의 테두리를 표면에 새겨넣은것처럼 만들어진다. 25행의 lineWidth()메쏘드는 중심에 오목 들어간 곬을 만드는 두 선분의 매개 화소너비를 지정한다. 26행의 setMidLine()메쏘드는 선의 중심에 있는 곬의 너비를 지정한다. 27행과 28행에서 setAlignment()와 setMargin()호출은 본문을 중심에 배치하고 본문과 틀사이에 8화소경계를 준다.

31～37행은 Raised그림자와 Box형태를 가지는 다른 표식자를 만든다. 이것은 마치도직4각형의 테두리가 표면에 솟아나게 한다.

이 틀에 그려지는 선들은 33행에서 setLineWidth()호출이 마루의 경계들이 1화소너비라는것을 지정하므로 앞의것보다 너비가 좁다.

39～43행은 직4각형이 솟아나오는 WinPanel형식의 표식자를 만든다. 너비는 기정값으로 설정되고 곬이나 마루가 없으므로 중간선의 너비는 없다.

45～50행은 Panel형식의 표식자를 만든다. 그것은 표식자전체가 표면아래로 가라앉는것처럼 보인다. 직4각형 테두리에 그려지는 선은 48행에서 setLineWidth()호출에 의해 4화소너비로 된다.

Qt와 KDE의 대다수 창문부품들은 기초클라스들로서 QFrame를 사용한다. 그 일부는 기정으로 틀을 현시하지만 그것들은 모두 틀을 현시할 능력이 있다. 다음 창문부품들중 임의의 것에서 메쏘드 setFrameStyle()호출은 틀이 나타나게 한다.

KAboutContainer	KMenuBar	QHButtonGroup
KAboutContributor	KMultiLineEdit	QHGroupBox

KAccelMenu	KMultiWallpaperList	QIconView
KApplicationTree	KPopupMenu	QIconView
KBackgroundDockWidget	KProgress	QLCDNumber
KCharSelect	KRuler	QLabel
KCharSelectTable	KSeparator	QListBox
KColorCells	KSplitList	QListView
KColorPatch	KStatusBar	QMenuBar
KDMView	KStatusBarLabel	QMultiLineEdit
KDatePicker	KTabListBoxTable	QPopupFrame
KDateTable	KTextBrowser	QPopupMenu
KDesktop	KThemeListBox	QProgressBar
KDockWindow	KToolBar	QScrollView
KEdit	KURLLabel	QSpinBox
KEyesWidget	KfindWindow	QSplitter
KFileSimpleView	KiKbdButton	QTableView
KFormulaToolBar	KiKbdMapInfoWidget	QTextBrowser
KGroupBox	KickerClientMenu	QTextEdit
KHTMLWidget	QButtonGroup	QTextView
KIOListView	QCanvasView	QVBox
KIconLoaderCanvas	QFileListBox	QVButtonGroup
KIconStyle	QFileListView	QVGroupBox
KImageTrackLabel	QGrid	QWellArray
KIntSpinBox	QGroupBox	QWidgetStack
	QHBox	

제6절. 틀을 만들기 위한 선택들

틀의 형태를 지정하는데 많은 환경설정값들을 사용할수 있다. 즉 Box, Panel, WinPanel, Hline 혹은 Vline이다. 또한 매개 형식은 솟아오르고 움푹 패여들어가고 또는 평평한것처럼 보이도록 설정될수 있다. 그리고 선의 너비를 지정할수 있다. 다음 실례는 틀을 구성하는 각이한 방법을 보여준다.

1. Box QFrame

boxframe이라는 프로그람은 Box형의 각이한 형태를 보여준다. 3가지 가능한 조절은 선너비, 중간선너비 그리고 형태가 솟아오르는가, 움푹 패이는가, 평평한가 하는것이다. 그림 7-11은 선의 너비가 1～3에서 변하고 중간선너비가 0～2에서 변하는 틀들의 형태를 보여준다.
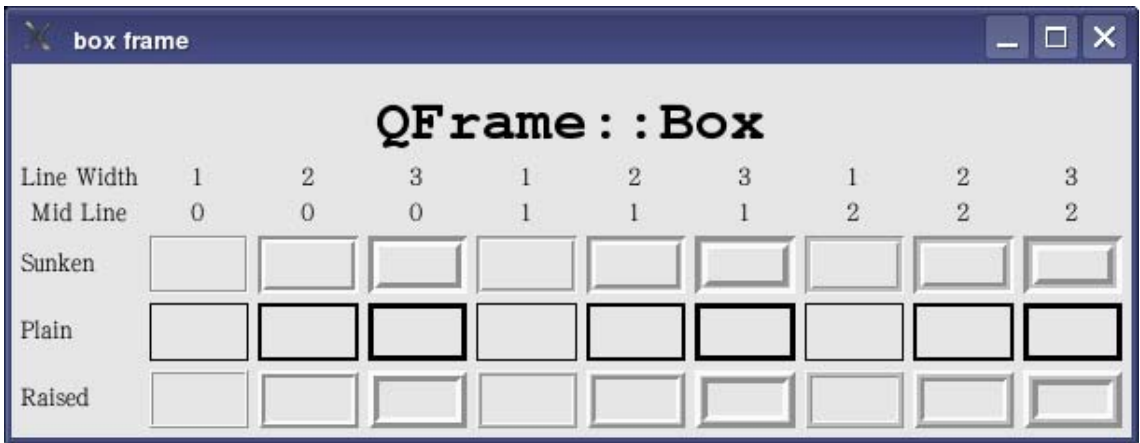
그림 7-11. Box QFrame형의 27가지 각이한 형태

다음 프로그람은 그림 7-11에 보여주는 틀들을 만드는데 사용된다. 그것은 살창배치를 사용하여 모든 틀과 표식자들을 배치하고 각이한 환경설정을 가지는 틀을 만들어 삽입하는 순환을 가지고있다.

```
1 /* boxframe.cpp */
2 #include <qlayout.h>
3 #include <qframe.h>
4 #include <qlabel.h>
5 #include <kapplication.h>
6 #include "boxframe.h"
7
8 int main(int argc, char **argv)
9 {
10     KApplication app(argc, argv,  "boxframe");
11     BoxFrame *boxframe = new BoxFrame();
12     boxframe->show();
13     app.setMainWidget(boxframe);
14     return(app.exec());
15 }
16
17 BoxFrame::BoxFrame(QWidget *parent,const char *name)
18     : QWidget(parent,name)
19 {
20     QLabel *label;
21     QFrame *frame;
```

```
22    QGridLayout *layout = new QGridLayout(this,6,10,5);
23
24    label = new QLabel("QFrame::Box",this);
25    label->setFont(QFont("Courier",24,QFont::Bold));
26    label->setAlignment(Qt::AlignHCenter);
27    layout->addMultiCellWidget(label,0,0,0,9);
28
29    label = new QLabel("Line Width",this);
30    label->setAlignment(Qt::AlignHCenter);
31    layout->addWidget(label,1,0);
32    label = new QLabel("Mid Line",this);
33    label->setAlignment(Qt::AlignHCenter);
34    layout->addWidget(label,2,0);
35    label = new QLabel("Sunken",this);
36    layout->addWidget(label,3,0);
37    layout->setRowStretch(3,1);
38    label = new QLabel("Plain",this);
39    layout->addWidget(label,4,0);
40    layout->setRowStretch(4,1);
41    label = new QLabel("Raised",this);
42    layout->addWidget(label,5,0);
43    layout->setRowStretch(5,1);
44
45    for(int i=0; i<9; i++) {
46            int lineWidth = (i % 3) + 1;
47            int midLineWidth = i / 3;
48            label = new QLabel(
49            QString("%1").arg(lineWidth),this);
50            label->setAlignment(Qt::AlignHCenter);
51            layout->addWidget(label,1,i+1);
52            label = new QLabel(
53                    QString("%1").arg(midLineWidth),this);
54            label->setAlignment(Qt::AlignHCenter);
55            layout->addWidget(label,2,i+1);
56
```

```
57          frame = new QFrame(this);
58          frame->setFrameStyle(QFrame::Box | QFrame::Sunken);
59          frame->setLineWidth(lineWidth);
60          frame->setMidLineWidth(midLineWidth);
61          layout->addWidget(frame,3,i+1);
62
63          frame = new QFrame(this);
64          frame->setFrameStyle(QFrame::Box | QFrame::Plain);
65          frame->setLineWidth(lineWidth);
66          frame->setMidLineWidth(midLineWidth);
67          layout->addWidget(frame,4,i+1);
68
69          frame = new QFrame(this);
70          frame->setFrameStyle(QFrame::Box | QFrame::Raised);
71          frame->setLineWidth(lineWidth);
72          frame->setMidLineWidth(midLineWidth);
73          layout->addWidget(frame,5,i+1);
74     }
75
76     resize(600,200);
77     layout->activate();
78 }
```

창문꼭대기의 표식자와 왼쪽에 있는 표식자들이 모두 만들어지고 24～27행에서 살창배치에 보관된다.

45행에서 시작하는 순환고리는 9개 렬의 틀을 현시하여야 하므로 9번 반복한다. 46행과 47행은 순환값을 사용함으로써 lineWidth와 midLineWidth값들을 계산한다. 현재렬의 두께를 2로 현시하기 위한 표식자들이 48～55행에서 만들어진다. 렬안의 3개 틀은 57～73행에서 만들어진다. setFrameStyle()의 3개호출은 모두 QFrame::Box형식을 사용하지만 각이한 그림자견본들이 주어진다. 매개 틀의 선두께는 계산된 값으로 설정된다.

이 절에서 나머지 실례들은 같은 기본코드를 사용하여 각이한 형식의 옵션들을 현시한다. 그러나 그림에서 알수 있는것처럼 임의의 형식에 사용할수 있는 옵션은 변화한다.

2. Panel QFrame

panelframe프로그람은 그림 7-12와 같이 Panel틀형태의 창문을 현시한다. 틀은 그 닫긴 령역이 표면우로 도드라지거나 아래로 움푹 패워들어가도록 할수 있는 하나의 직선으로 만든다. 이전의 실례에서 사용한 중간선값은 틀이 하나의 직선으로 구성되므로 여기에 영향을

주지 않는다.
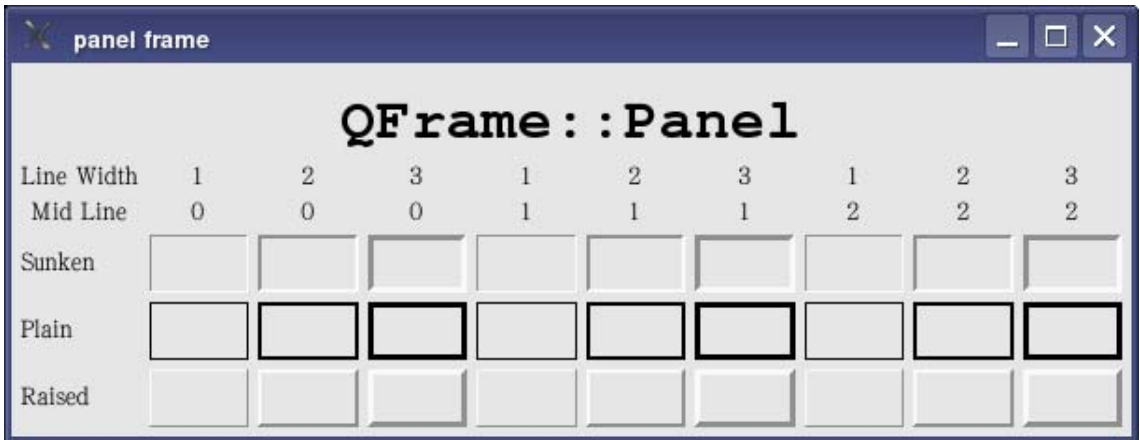
그림 7-12. 각이한 선두께를 가지는 3가지 형식의 Panel틀

3. WinPanel QFrame

프로그람 winpanelframe은 그림 7-13에 보여주는 창문을 현시한다. WinPanel틀의 모양은 직선너비가 2화소로 설정된 Panel틀과 같다. WinPanel에서 선두께는 변경할수 없다. 이 형식의 틀은 원래 Windows조작체계의 모양을 본딴것이다.
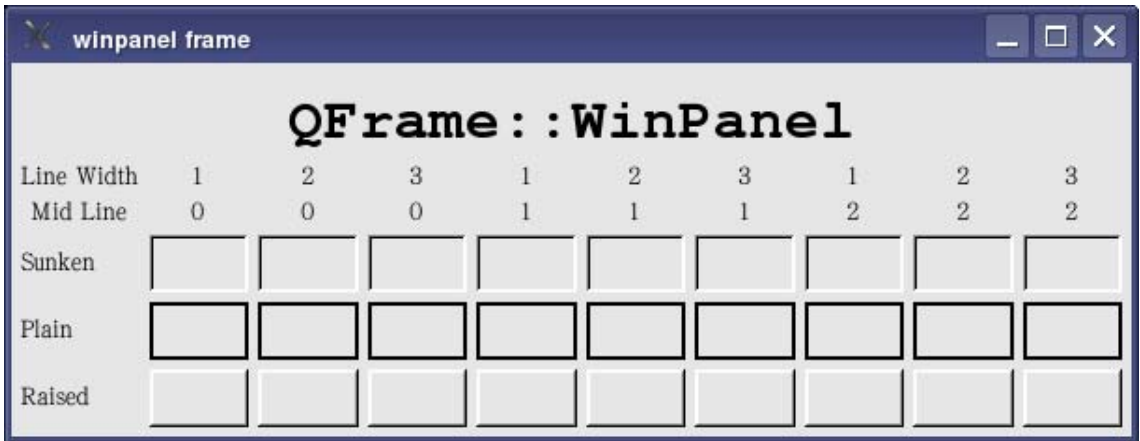
그림 7-13. 3가지 형식의 WinPanel틀

4. QFrame에 의한 직선그리기

2개의 QFrame형식은 틀이 아니라 직선이다. HLine형식을 사용하여 QFrame에 그림 7-14와 같이 수평선을 그릴수 있다. Box형태에 적용할수 있는 모든 변경은 HLine에도 적용할수 있다. 즉 매개 선은 3행으로 그려지는데 중앙선의 두께는 지정된 중간선값을 가지고 다른 두 선의 너비는 지정된 선두께값을 가진다. 선이 표면우로 올리솟거나 움푹 들어간것처럼 보이도록 그늘이 설정된다.
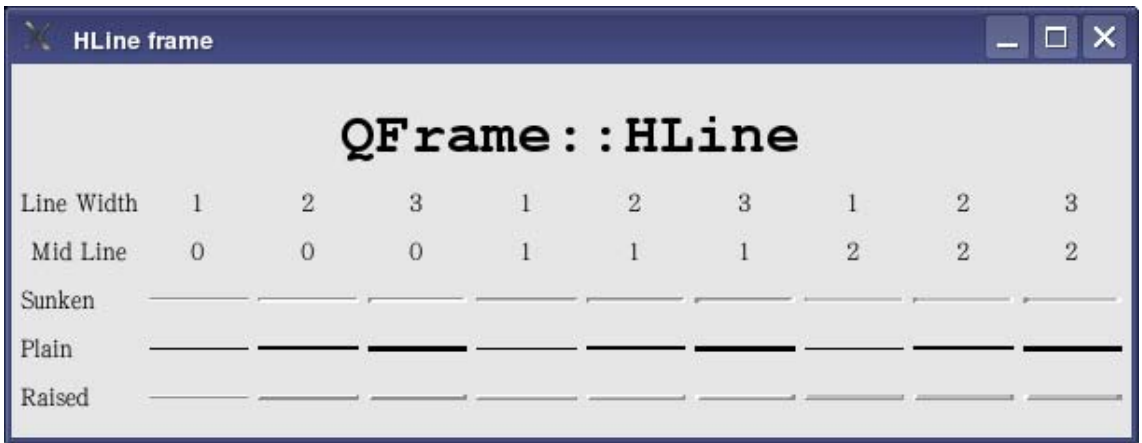
그림 7-14. 수평선으로 그린 QFrame

그림 7-15와 같이 수직선을 그릴수도 있다. 수직선은 수평선과 같은 옵션들을 사용하여 구성할수 있다.
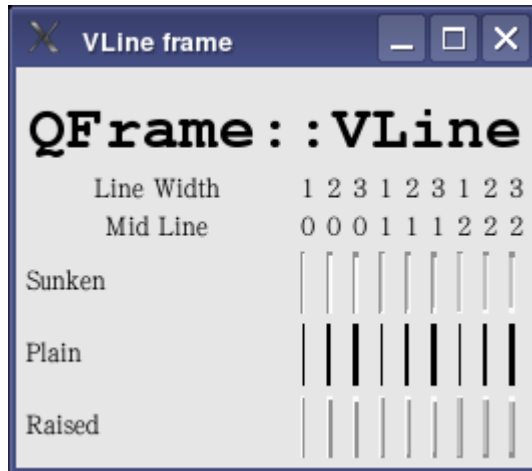
그림 7-15. 수직선으로 그려진 QFrame

제7절. 창문의 실제상태공유

QSplitter창문부품을 리용하면 창문에 1개 이상의 창문부품을 현시하고 개별적인 창문부품들이 겹치지 않도록 사용자가 크기를 변경할수 있다. 그림 7-16은 한쌍의 본문편집창문을 포함하는데 사용되고있는 분할기를 보여준다.
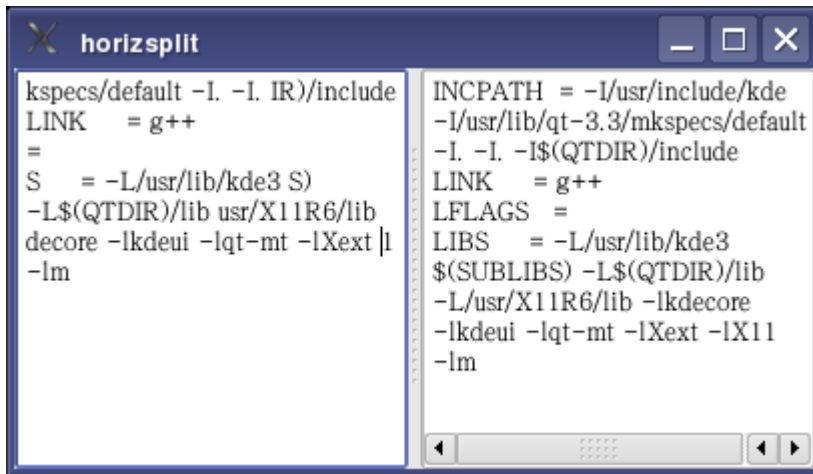
그림 7-16. 2개의 편집창문을 포함하는 QSplitter

2개 편집구획사이의 띠는 이동할수 있고 한 창문부품의 너비가 커질 때 다른 창문부품의 크기도 커진다. 이 실례에서 편집창문부품들은 현시하고있는 본문이 자동적으로 행바꾸어지도록 설정되므로 그림 7-17처럼 띠를 왼쪽으로 이동하면 두 편집창문부품들의 크기가 변경되고 본문은 스스로 재배렬된다.
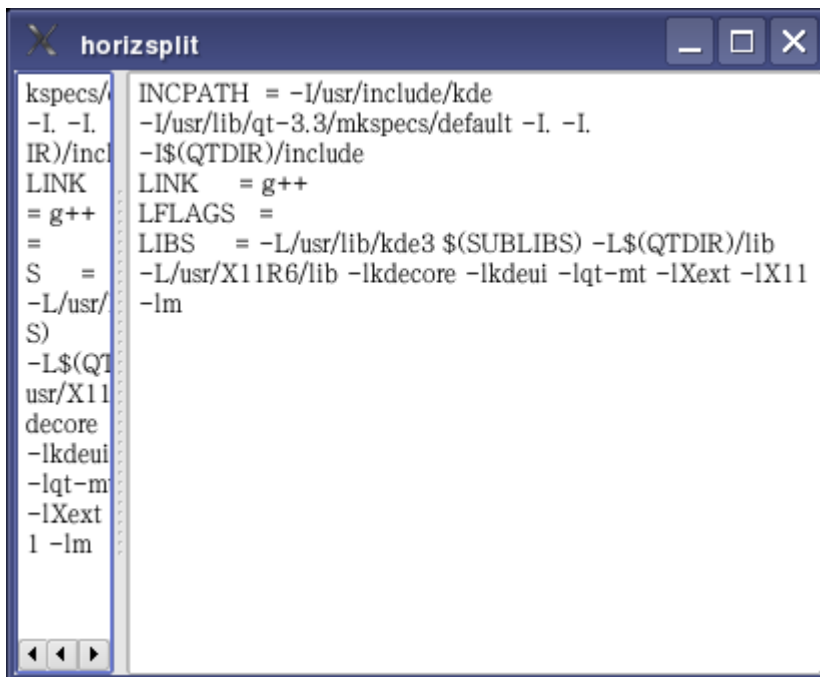
그림 7-17. 2개 편집창문의 크기를 변경한 결과를 보여주는 QSplitter

다음 프로그람은 그림 7-16과 그림 7-17에서 현시하는 창문들을 만든다.

HorizSplit머리부파일

```
1 /* horizsplit.h */
2 #ifndef HORIZSPLIT_H
```

```
3 #define HORIZSPLIT_H
4
5 #include <qsplitter.h>
6
7 class HorizSplit: public QSplitter
8 {
9     Q_OBJECT
10 public:
11     HorizSplit(QWidget *parent=0,const char *name=0);
12 };
13
14 #endif
```

이 머리부파일은 QSplitter를 계승하는 클라스 HorizSplit를 선언한다.

HorizSplit

```
1 /* horizsplit.cpp */
2 #include <kapplication.h>
3 #include <qmultilineedit.h>
4 #include "horizsplit.h"
5
6 int main(int argc, char **argv)
7 {
8     KApplication app(argc, argv,  "horizsplit");
9     HorizSplit *horizsplit = new HorizSplit();
10    horizsplit->show();
11    app.setMainWidget(horizsplit);
12    return(app.exec());
13 }
14
15 HorizSplit::HorizSplit(QWidget *parent,const char *name)
16    : QSplitter(parent,name)
17 {
18    QMultiLineEdit *leftEdit = new QMultiLineEdit(this);
19    leftEdit->setMinimumWidth(50);
20    leftEdit->setWordWrap(QMultiLineEdit::WidgetWidth);
21
```

```
22    QMultiLineEdit *rightEdit = new QMultiLineEdit(this);
23    rightEdit->setMinimumWidth(50);
24    rightEdit->setWordWrap(QMultiLineEdit::WidgetWidth);
25
26    resize(400,200);
27 }
```

분할창문의 설정과정은 간단히 분할기안의 매개 구획들에 창문부품을 하나씩 삽입하는 과정이다. 15행에서 시작되는 구성자는 이것을 부모클라스로 사용하는 2개의 QMultiLineEdit 객체를 만든다.(this는 현재객체에 대한 참고이다.) 또한 HorizSplit클라스는QSplitter클라스이고 QSplitter는 개별적인 구획들안의 자식창문부품들을 모두 관리하므로 그밖에 어떤 조작도 필요하지 않다. 본문편집창문의 최소허용너비는 분할띠의 이동에 적은 제한을 주기 위하여 50으로 설정되며 만일 아래한계를 주지 않으면 띠는 한 창문이 완전히 보이지 않을 때가지 이동될수 있다.

QSplitter의 기정값은 수평으로 창문부품들을 배렬하는것이다. 다음 실례는 수직으로 창문부품을 배렬한다.

VertSplit.

```
1 /* vertsplit.cpp */
2 #include <kapplication.h>
3 #include <qmultilineedit.h>
4 #include "vertsplit.h"
5
6 int main(int argc, char **argv)
7 {
8     KApplication app(argc, argv,  "vertsplit");
9     VertSplit *vertsplit = new VertSplit();
10    vertsplit->show();
11    app.setMainWidget(vertsplit);
12    return(app.exec());
13 }
14
15 VertSplit::VertSplit(QWidget *parent,const char *name)
16    : QSplitter(parent,name)
17 {
18    setOrientation(Vertical);
19
```

```
20    QMultiLineEdit *topEdit = new QMultiLineEdit(this);
21    topEdit->setMinimumHeight(50);
22    topEdit->setWordWrap(QMultiLineEdit::WidgetWidth);
23
24    QMultiLineEdit *middleEdit = new QMultiLineEdit(this);
25    middleEdit->setMinimumHeight(50);
26    middleEdit->setWordWrap(QMultiLineEdit::WidgetWidth);
27
28    QMultiLineEdit *bottomEdit = new QMultiLineEdit(this);
29    bottomEdit->setMinimumHeight(50);
30    bottomEdit->setWordWrap(QMultiLineEdit::WidgetWidth);
31
32    resize(200,400);
33 }
```

18행의 setOrientation()호출은 그 창문부품들을 겹쳐서 배렬하도록 QSplitter에게 지시한다. 처음에 추가된것은 꼭대기에 있다. 분할기에 많은 창문부품을 추가할수 있는데 이 실례에서는 3개의 창문부품을 포함하므로 그것들을 분리하는 2개 띠를 가진다. 결과를 그림 7-18에 보여준다.
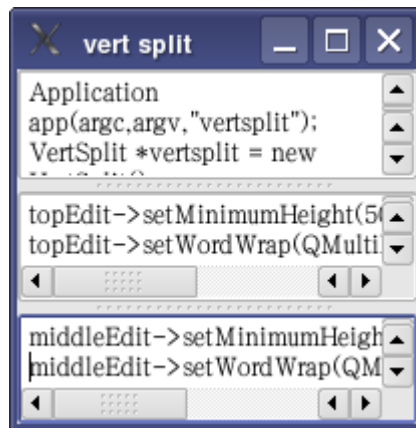
그림 7-18. 3개 편집창문을 포함하는 수직QSplitter

요 약

이 장에서는 응용프로그람과 창문을 보는 사용자에게 어떤 의미를 주도록 창문부품들을 구성하는데 사용할수 있는 몇가지 기본방법을 설명하였다. 이 장은 다음과 같은 내용을 설명하였다.

- 특수창문부품을 사용하여 단추그룹의 현시를 조직하고 그룹안의 모든 단추가 같은 처리부를 사용하여 마우스찰칵을 알릴수 있다.

- 라지오단추그룹은 한번에 1개 단추만 선택되도록 물리적으로 제한하여 서로 련결할수 있어야 한다.
- 검사단추(절환단추)들은 물리적으로 서로 련결되지 않고 자주 론리적으로 서로 련결되며 그룹으로서 고찰되여야 한다.
- 장식틀은 창문부품주위에 그릴수 있다. 장식보다도 틀은 혼돈되기 쉬운 현시에서 창문부품관계를 명백히 하는데 사용될수 있다.
- 창문부품집합은 같은 공간을 공유하고 사용자가 창문부품들사이에 앞뒤로 이동하게 할수 있다.

제8장. 마우스와 건반

학습내용
하드웨어로부터 응용프로그람에로 사건을 전달하기
마우스자료의 읽어들이기
마우스와 그 유표의 조종
건반으로부터 들어오는 사건의 해석

표준KDE/Qt응용프로그람은 사용자로부터 소식을 들을 때까지 아무것도 하지 않는다. 응용프로그람이 처음으로 실행될 때 필요한 초기화를 하고 창문을 현시한 다음 정지상태에 들어가서 사용자의 조작을 기다린다. 대다수의 경우에 프로그람은 마우스 또는 건반으로부터 입력을 기다리지만 입력은 빛펜, 마우스바퀴, 도형타블레트, 추적볼 혹은 다른 입력장치로부터도 진행될수 있다. 대부분의 응용프로그람은 창문부품의 부분으로서 포함된 미리 정의된 신호와 처리부들의 모임을 사용하여 작성될수 있으나 들어오는 사건흐름에 자기 프로그람을 직접 련결할 필요가 가끔 제기된다. 또한 만약 자체의 창문부품을 만들려고 한다면 들어오는 사건들을 신호로 변환하는 방법을 알고있어야 한다.

제1절. 포구로부터 처리부에로

먼저 사건의 생명주기에 대하여 간단히 설명한다.

사건발생은 하드웨어로부터 시작한다. 마우스가 새 위치로 이동하고 건반의 건을 놓고 마우스단추를 누르거나 건이 눌리워져있다고 하자. 사건을 발생시키는 장치는 물리적으로 콤퓨터에 련결되므로 사건은 새치기를 일으키고 구동프로그람으로 알려진 작은 프로그람은 포구로부터 정보를 읽어들인다. 장치구동프로그람의 주요한 일감은 하드웨어사건을 쏘프트웨어사건으로 변환하는것이다.

장치구동프로그람은 특정한 포구에서 사건을 기다리도록 되여야 한다. 이것은 응용프로그람이 파일을 여는것과 거의 같은 방법으로 포구를 열 때 프로그람으로부터 수행된다. 포구들은 모두 /dev등록부에 있고 마치도 파일인것처럼 프로그람에 의해서 보낼수 있다. 프로그람이 포구(즉 /dev등록부안의 이름)를 열 때 Linux핵심은 장치구동프로그람을 선택하고 포구와 응용프로그람사이에 정보를 넘기는 일감을 준다.

Qt와 KDE에서 구동프로그람으로부터 사건을 받아들이는 응용프로그람은 X창문체계이다. 장치구동프로그람으로부터 오는 매개 사건은 XEvent라는 특별한 내부형식으로 형식화된다. 창문관리기는 XEvent를 검사하여 그 목적창문을 결정한다. 실례로 사건이 건반찰칵이면 그것은 현재 초점을 가지는 창문에 전송된다. 그것이 마우스찰칵이라면 마우스지시기의 x와 y자리표들은 보통 그 창문을 결정한다. 어느 창문이 사건을 받아들이는가 알아냄으로써 창문관리기는 어느 응용프로그람이 사건을 받아들여야 하는가 결정할수 있다.

X창문프로그람이 사건을 처리하기 위하여 함수 XNextEvent()를 읽어들이는 저준위대기

렬을 계속 호출하도록 순환고리를 설정한다. 이 함수가 돌아올 때마다 대기렬로부터 하나의 사건을 받아들인다. 사건들은 일정한 형의 사건들을 취급하도록 할당된 메쏘드를 호출함으로써 응용프로그람에 발송된다.

기본함수에서 자기 프로그람이 QApplication::exec()나 KApplication::exec()를 호출할 때 그것은 XNextEvent()를 호출하는 련속적인 순환고리를 실행하여 하나의 XEvent를 읽어들여 Qt사건들중 하나로 변환하고 그것을 취급하는 적당한 메쏘드를 호출한다.

이 사건접수방법은 QWidget클라스의 부분으로 정의되며 가상 및 비공개로서 선언된다. 이 메쏘드들에 의해 수행된 작용은 극히 작으므로 QWidget의 파생클라스를 만들고 사건들을 받아들이는 메쏘드들을 재정의해야 한다. 실례로 QButton클라스는 mouseReleaseEvent()이라는 QWidget메쏘드를 재정의하여 QButton이 clicked()신호를 발생하도록 하게 할수 있다. 이것은 마우스가 단추를 동작시킬 때마다 응용프로그람이 clicked()사건을 받아들일수 있게 한다.

제2절. 마우스사건

마우스사건은 현재 마우스단추들을 누르면 마우스지시기의 현재 위치를 반영한다. QWidget클라스의 메쏘드는 마우스사건이 발생할 때마다 호출된다. 만일 자체로 마우스사건처리를 만들려고 한다면 QWidget에서 정의된 사건조종메쏘드들을 재정의한다.

다음의 프로그람은 마우스를 추적하고 단추, 건 및 마우스지시기의 위치들을 현시한다. 그림8-1과 같이 오른쪽의 빈 창문부품은 마우스이동을 감시하는 창문부품이고 매개 마우스작용은 왼쪽에 현시된다. 이 실례는 들어오는 사건을 신호로 변환하는 방법을 보여준다. 마우스사건이 들어올 때마다 그로부터 들어오는 자료는 설명문자렬을 만드는데 사용된다. 이 문자렬은 신호로서 발생되고 다른 클라스의 처리부에 의해서 받아들여진다.
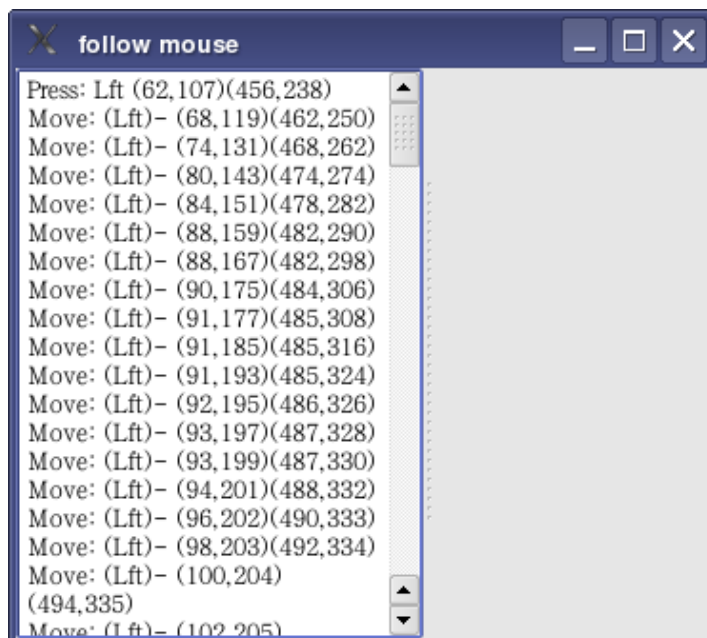
그림 8-1. 마우스동작을 추적하고 현시하는 프로그람

MouseSensor머리부파일

```
1 /* mousesensor.h */
2 #ifndef MOUSESENSOR_H
3 #define MOUSESENSOR_H
4
5 #include <qwidget.h>
6 #include <qevent.h>
7 #include <qstring.h>
8
9 class MouseSensor: public QWidget
10 {
11     Q_OBJECT
12 public:
13     MouseSensor(QWidget *parent=0,const char *name=0);
14 private:
15     void emitDescription(const QString &,QMouseEvent *);
16     virtual void mousePressEvent(QMouseEvent *event);
17     virtual void mouseReleaseEvent(QMouseEvent *event);
18     virtual void mouseDoubleClickEvent(QMouseEvent *event);
19     virtual void mouseMoveEvent(QMouseEvent *event);
20 signals:
21     void description(QString &);
22 };
23
24 #endif
```

MouseSensor클라스는 마우스동작을 추적하는 창문부품이다. 그것은 그림 8-1의 오른쪽에 빈 창틀(구획)로서 나타난다. 16～19행의 가상메쏘드들은 기초클라스 QWidget의 메쏘드들을 재정의하므로 모든 마우스사건들은 기초클라스가 아니라 가상메쏘드들에 들어온다.

MouseSensor

```
1 /* mousesensor.cpp */
2 #include <qstring.h>
3 #include "mousesensor.h"
4
5 MouseSensor::MouseSensor(QWidget *parent,const char *name)
6       : QWidget(parent,name)
```

```
7 {
8     setMinimumSize(300,300);
9 }
10 void MouseSensor::mousePressEvent(QMouseEvent *event)
11 {
12     emitDescription(QString("Press: "),event);
13 }
14 void MouseSensor::mouseReleaseEvent(QMouseEvent *event)
15 {
16     emitDescription(QString("Release: "),event);
17 }
18 void MouseSensor::mouseDoubleClickEvent(QMouseEvent *event)
19 {
20     emitDescription(QString("DoubleClick: "),event);
21 }
22 void MouseSensor::mouseMoveEvent(QMouseEvent *event)
23 {
24     emitDescription(QString("Move: "),event);
25 }
26 void MouseSensor::emitDescription(const QString &typeStr,
27     QMouseEvent *event)
28 {
29     QString btnStr(typeStr);
30     ButtonState state = event->state();
31     if(state & ControlButton)
32             btnStr+= "Ctl-";
33     if(state & AltButton)
34             btnStr+= "Alt-";
35     if(state & ShiftButton)
36             btnStr+= "Shft-";
37     if(state & LeftButton)
38             btnStr += "(Lft)-";
39     if(state & MidButton)
40             btnStr += "(Mid)-";
41     if(state & RightButton)
```

```
42          btnStr += "(Rgt)-";
43    ButtonState button = event->button();
44    if(button & LeftButton)
45          btnStr += "Lft";
46    if(button & MidButton)
47          btnStr += "Mid";
48    if(button & RightButton)
49          btnStr += "Rgt";
50
51    QString str = QString("%1 (%2,%3)(%4,%5) ")
52          .arg(btnStr)
53          .arg(event->x()).arg(event->y())
54          .arg(event->globalX()).arg(event->globalY());
55
56    emit description(str);
57 }
```

MouseSensor클라스는 QWidget를 기초클라스로 사용하므로 창문부품이다. 이 클라스에서 마우스사건들을 받아들이는 4개 메쏘드는 기초클라스의 4개 가상메쏘드를 재정의한다. 메쏘드 mousePressEvent()는 마우스단추들중 하나를 누를 때마다 호출되며 메쏘드 mouseReleaseEvent()는 마우스단추들중 하나를 놓을 때마다 호출된다. 메쏘드 mouseDoubleClick()는 마우스단추를 일정한 시간간격으로 두번 눌렀다 놓을 때마다 호출된다. 이것은 두번찰칵작용이 5개의 사건 즉 2개의 단추누르기사건, 2개의 단추놓기사건 그리고 1개의 두번찰칵사건을 생성한다는것을 의미한다.

22행의 메쏘드 mouseMoveEvent()는 마우스가 창문안의 새로운 위치로 이동될 때마다 호출된다. 2가지 조작방법이 있다. 그 방법은 매개 마우스이동에 대하여 호출되거나 마우스단추를 누르고 이동할 때만 호출될수 있다. 기정은 단추들중의 하나를 누르고있을것을 요구하지만 메쏘드 setMouseTracking(TRUE)호출은 마우스위치를 항상 통보한다.

26행에서 시작하는 emitDescription()은 서술문자렬을 만들고 문자렬을 포함하는 신호를 보내는 모든 사건수신메쏘드들에 의해 사용된다. 정보는 모두 QMouseEvent객체안에 있다.

ButtonState값은 30행에서 사건의 state()메쏘드를 호출함으로써 얻어진다. 이 변수는 마우스사건이 발생한 시간에 누르고있는 단추들(있다면)을 표시하는 기발들을 보관한다. 6개의 가능한 단추를 설정할수 있다. 즉 Alt, Ctrl, 그리고 Shift와 함께 3개 마우스단추. 43행의 button()호출에 의해 얻어진 ButtonState값은 사건을 일으킨 단추(있다면)이다. 실례로 Ctrl건과 오른쪽 마우스단추를 누르고있으면 왼쪽 마우스단추누르기는 Ctrl건과 오른쪽마우스단추지시기가 state()호출로 되돌아오게 하고 왼쪽 마우스단추지시기는 button()으로 되돌아오게 한다.

알아두기: Alt건은 ButtonState의 기발값에 의해 표시되며 Alt건은 실제로 마우스사건에서 통보되지 않는다. 이것은 KDE가 Alt건을 마우스와 함께 사용할 때 Alt건을 받아들이기때문이다. Alt-왼쪽마우스단추는 창문을 이동하는데 사용되고 Alt-중간단추는 초점을 다음 창문으로 이동하며 Alt-right단추는 현재 창문을 다시 조절한다. 2단추마우스를 사용하면 대체로 동시에 두 단추를 누름으로써 가운데 단추를 모의할수 있고 Alt건없는 건반은 일반적으로 같은 동작을 하는 메타건을 가지고있다.

QMouseEvent객체에 2개의 마우스위치가 있다. 하나는 전체창문의 왼쪽웃구석에 관한 x와 y자리표를 표시하며 다른것은 현재 창문의 왼쪽웃구석의 x와 y값을 표시한다. 51행에서 QString객체는 사건의 설명을 보유하기 위해 만든다. 53행과 54행은 사건을 설명하는 문자렬로서 자리표들을 형식화한다. 대체로 국부자리표들은 자기가 바라는것이지만 가끔 마우스의 대역(세계)위치를 알아야 한다.

56행의 emit문은 들어오는 사건을 나가는 신호로 변환하는 마지막 단계이다. 이러한 부류의 처리는 실례로 마우스단추사건을 click()라는 이름의 신호로 변환하는 QPushButton에서 발생한다. 이 실례에서 사건의 설명을 담고있는 QString이 포함된 신호가 발생된다.

FollowMouse머리부파일

```
1 /* followmouse.h */
2 #ifndef FOLLOWMOUSE_H
3 #define FOLLOWMOUSE_H
4
5 #include <qsplitter.h>
6 #include <qstring.h>
7 #include <qmultilineedit.h>
8
9 class FollowMouse: public QSplitter
10 {
11     Q_OBJECT
12 public:
13     FollowMouse(QWidget *parent=0,const char *name=0);
14 public slots:
15     void newline(QString &);
16 private:
17     QMultiLineEdit *edit;
18 };
19
20 #endif
```

FollowMouse클라스는 그림 8-1에서 이미 보여준 최상위창문이다. 그것은 왼쪽의 QMultiLineEdit객체와 오른쪽의 MouseSensor객체를 모두 관리하는 QSplitter에 기초하고있다. FollowMouse

```
1 /* followmouse.cpp */
2 #include <kapplication.h>
3 #include <qstring.h>
4 #include "mousesensor.h"
5 #include "followmouse.h"
6
7 int main(int argc, char **argv)
8 {
9     KApplication app(argc, argv, "followmouse");
10    FollowMouse *followmouse = new FollowMouse();
11    followmouse->show();
12    app.setMainWidget(followmouse);
13    return(app.exec());
14 }
15
16 FollowMouse::FollowMouse(QWidget *parent,const char *name)
17    : QSplitter(parent,name)
18 {
19    edit = new QMultiLineEdit(this);
20    edit->setMinimumWidth(80);
21    edit->setReadOnly(TRUE);
22
23    MouseSensor *sensor = new MouseSensor(this);
24    sensor->setMinimumWidth(80);
25
26    connect(sensor,SIGNAL(description(QString &)),
27            this,SLOT(newline(QString &)));
28
29    resize(10,10);
30 }
31 void FollowMouse::newline(QString &str)
32 {
```

```
33    edit->insertLine(str);
34    edit->setCursorPosition(5000,0);
35 }
```

16행에서 시작하는 FollowMouse구성자는 19～21행에서 QMultiLineEdit창문부품을 만든다. setReadOnly()호출은 편집을 금지하고 편집기를 현시만 할수 있는 본문창문으로 만든다. 편집기창문부품이 처음으로 만들어지므로 그것이 왼쪽에 나타난다. 23행에서 만들어진 MouseSensor는 오른쪽에 나타난다.

이 클라스가 마우스사건서술을 받아들이기 위하여 26행에서 connect()호출을 만들고 FollowMouse의 description()신호로부터 MouseSensor의 처리부 newLine()에로의 련결을 확립한다.

31행의 처리부메쏘드는 모든 마우스사건의 서술문자렬과 함께 호출된다. 33행의 insertLine()호출은 QMultiLineEdit창문부품에 현시된 본문의 아래에 문자렬을 추가한다. setCursorPos()호출은 가장 새로운 문자렬(바닥에 있는것)이 보인다는것을 담보한다. 목록의 실제성원수를 넘는 첨수를 지정하면 QMultiLineEdit이 마지막것을 선택한다.

제3절. 마우스의 포획과 해방

응용프로그람에는 마우스를 조종할수 있는 창문부품이 있다. 이것은 마우스지시기의 이동은 제한하지 않지만 다른 모든 창문부품들(이것 또는 다른 응용프로그람안의)이 마우스사건을 받아들이는것을 금지한다.

경고 마우스를 포획하면 또한 해방기구가 있는가 확인하여야 한다. 자기의 프로그람이 마우스를 포획한 다음 해방하지 않으면 마우스는 잠그어진다. 즉 건반은 동작하지만 마우스는 자기의 프로그람이 끝날 때까지 사용금지된다.

다음 프로그람은 그림 8-2에서 보여준것과 같은 창문을 현시한다. 꼭대기단추는 마우스를 포획하고 유표를 십자모양으로 변경한다. 아래 단추는 마우스를 해방한다.
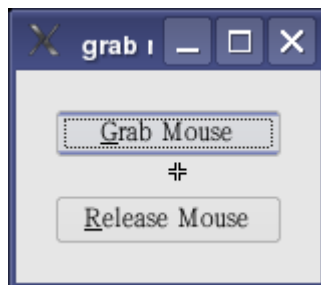
그림 8-2. 마우스의 포획

GrabMouse머리부파일

```
1 /* grabmouse.h */
2 #ifndef GRABMOUSE_H
3 #define GRABMOUSE_H
4
```

```
5 #include <qwidget.h>
6 #include <qlayout.h>
7 #include <qpushbutton.h>
8
9 class GrabMouse: public QWidget
10 {
11     Q_OBJECT
12 public:
13     GrabMouse(QWidget *parent=0,const char *name=0);
14 private:
15     QPushButton *grabButton;
16     QPushButton *relButton;
17 public slots:
18     void mouse_grab();
19     void mouse_release();
20 };
21
22 #endif
```

GrabMouse

```
1 /* grabmouse.cpp */
2 #include <kapplication.h>
3 #include <qcursor.h>
4 #include "grabmouse.h"
5
6 int main(int argc, char **argv)
7 {
8     KApplication app(argc, argv, "grabmouse");
9     GrabMouse *grabmouse = new GrabMouse();
10    grabmouse->show();
11    app.setMainWidget(grabmouse);
12    return(app.exec());
13 }
14
15 GrabMouse::GrabMouse(QWidget *parent,const char *name)
```

```
16    : QWidget(parent,name)
17 {
18    QVBoxLayout *layout = new QVBoxLayout(this,20);
19
20    grabButton = new QPushButton("Grab Mouse",this);
21    grabButton->setMinimumSize(grabButton->sizeHint());
22    layout->addWidget(grabButton);
23    connect(grabButton,SIGNAL(clicked()),
24            this,SLOT(mouse_grab()));
25
26    relButton = new QPushButton("Release Mouse",this);
27    relButton->setMinimumSize(relButton->sizeHint());
28    layout->addWidget(relButton);
29    connect(relButton,SIGNAL(clicked()),
30            this,SLOT(mouse_release()));
31
32    resize(10,10);
33    layout->activate();
34 }
35 void GrabMouse::mouse_grab()
36 {
37    relButton->grabMouse(QCursor(CrossCursor));
38 }
39 void GrabMouse::mouse_release()
40 {
41    relButton->releaseMouse();
42 }
```

GrabMouse클라스는 18행에서 한쌍의 단추들을 가진 수직칸을 사용하는 창문부품이다. grabButton이라는 우에 있는 단추는 35행에서 정의된 처리부메쏘드 mouse_grab()에 련결된 clicked()신호를 가지고있다. 마찬가지로 relButton은 39행에서 정의된 처리부메쏘드 mouse_released()에 련결된다.

우의 단추가 선택될 때마다 relButton의 grabMouse()메쏘드가 호출되여 relButton이 마우스를 포획하게 한다. grabMouse()에 유표정의를 넘기면 포획기간에 유표모양이 변경되게 한다. 그것은 포획을 실행한 relButton이므로 마우스에 응답할 유일한 창문부품이며 체계에 마우스조종을 돌려보내기 위하여 releaseMouse()를 호출한다.

제4절. 유표의 형태변경

내부유표의 표준모임을 사용하여 프로그람의 현재상태를 사용자에게 통지할수 있다. 유표를 변경하는 메쏘드는 QWidget에 있으므로 어떠한 현시가능객체라도 변경된 유표를 가질수 있다.

유표의 모양을 바꿀 때 변경은 오직 자기가 지정한 구역에 적용한다. 실례로 자기 응용프로그람의 최상위창문을 위한 유표를 변경한다면 그것은 제목띠를 변경하지 않지만 자기의 최상위창문의 자식이나 또는 자손인 모든 창문부품들을 변경한다. 그러나 자식창문부품들중의 하나가 자기 유표설정을 가지고있다면 그것(과 그아래의 모든 자손)은 자기의 유표를 가지고있다.

다음 실례는 모든 표준유표중에서 동적으로 선택할수 있게 한다. 전체 창문에 한개 유표를 적용하고 창문내부의 한개 단추에 다른 유표를 적용할수 있다. 그림 8-3에 보여준것처럼 유표들의 이름은 왼쪽에 렬거된다. 유표이름이 선택될 때마다 그것은 전체 창문에 대하여 기정유표로 된다. 오른쪽에 있는 Select단추를 사용하면 현재 선택한 유표가 그 자체의 개별적인 유표로서 할당되도록 한다. 즉 그의 유표가 더는 그 부모로부터 계승되지 않는다.
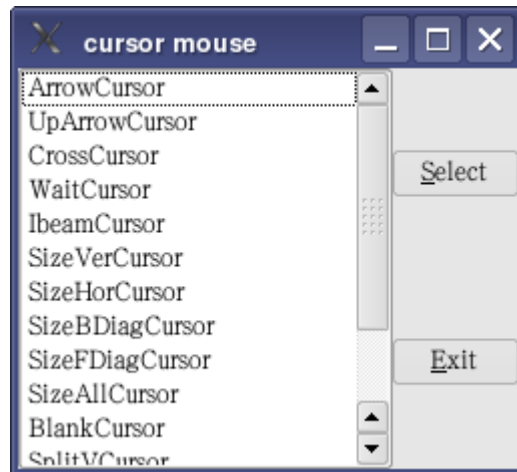
그림 8-3. 유표선택목록

CursorMouse머리부파일

```
1 /* cursormouse.h */
2 #ifndef CURSORMOUSE_H
3 #define CURSORMOUSE_H
4
5 #include <qlayout.h>
6 #include <qpushbutton.h>
7
8 class CursorMouse: public QWidget
```

```
9 {
10     Q_OBJECT
11 public:
12     CursorMouse(QWidget *parent=0,const char *name=0);
13 private:
14     QPushButton *selectButton;
15     QPushButton *exitButton;
16     int cursorID;
17 public slots:
18     void changeCursor(int);
19     void selectCursor();
20     void shutdown();
21 };
22
23 #endif
```

CursorMouse클라스는 3개의 처리부를 포함한다. 그중 2개는 유표를 변경하는데 사용되고 3번째는 프로그람을 완료한다. 16행에 선언된 cursorID는 현재 선택된 유표의 ID번호를 보관한다.

CursorMouse

```
1 /* cursormouse.cpp */
2 #include <kapplication.h>
3 #include <qcursor.h>
4 #include <qlistbox.h>
5 #include "cursormouse.h"
6
7 struct cursStruct {
8     QString name;
9     int number;
10 } curs[] = {
11     { "ArrowCursor",ArrowCursor },
12     { "UpArrowCursor",UpArrowCursor },
13     { "CrossCursor",CrossCursor },
14     { "WaitCursor",WaitCursor },
15     { "IbeamCursor",IbeamCursor },
16     { "SizeVerCursor",SizeVerCursor },
```

```
17     { "SizeHorCursor",SizeHorCursor },
18     { "SizeBDiagCursor",SizeBDiagCursor },
19     { "SizeFDiagCursor",SizeFDiagCursor },
20     { "SizeAllCursor",SizeAllCursor },
21     { "BlankCursor",BlankCursor },
22     { "SplitVCursor",SplitVCursor },
23     { "SplitHCursor",SplitHCursor },
24     { "PointingHandCursor",PointingHandCursor },
25     { "BitmapCursor",BitmapCursor }
26 };
27
28 int main(int argc, char **argv)
29 {
30     KApplication app(argc, argv, "cursormouse");
31     CursorMouse *cursormouse = new CursorMouse();
32     cursormouse->show();
33     app.setMainWidget(cursormouse);
34     return  (app.exec());
35 }
36
37 CursorMouse::CursorMouse(QWidget *parent,const char *name)
38     : QWidget(parent,name)
39 {
40     QHBoxLayout *horlayout = new QHBoxLayout(this);
41
42     QListBox *list = new QListBox(this);
43     for(int i=0; i<sizeof(curs)/sizeof(cursStruct); i++)
44             list->insertItem(curs[i].name);
45     horlayout->addWidget(list);
46     connect(list,SIGNAL(highlighted(int)),
47             this,SLOT(changeCursor(int)));
48
49     QVBoxLayout *verlayout = new QVBoxLayout(30);
50
51     selectButton = new QPushButton("Select",this);
```

```
52    selectButton->setMinimumSize(selectButton->sizeHint());
53    verlayout->addWidget(selectButton);
54    connect(selectButton,SIGNAL(clicked()),
55          this,SLOT(selectCursor()));
56
57    exitButton = new QPushButton("Exit",this);
58    exitButton->setMinimumSize(exitButton->sizeHint());
59    verlayout->addWidget(exitButton);
60    connect(exitButton,SIGNAL(clicked()),
61          this,SLOT(shutdown()));
62
63    horlayout->addLayout(verlayout);
64
65    resize(250,200);
66    horlayout->activate();
67 }
68 void CursorMouse::changeCursor(int index)
69 {
70    cursorID = curs[index].number;
71    setCursor(QCursor(cursorID));
72 }
73 void CursorMouse::selectCursor()
74 {
75    selectButton->setCursor(QCursor(cursorID));
76 }
77 void CursorMouse::shutdown()
78 {
79    kapp->exit(0);
80 }
```

7～26행의 배렬은 미리 정의된 유표들의 이름과 ID번호들을 보관한다. 이름들은 목록칸에서 선택본문으로 사용되며 ID값들은 QCursor의 구성자에서 인수로 사용된다.

37행에서 선언되는 CursorMouse클라스는 응용프로그람의 최상위창문부품이다. 그것은 그림 8-3에 보여준것처럼 수평칸을 리용하여 왼쪽에 QListBox를 보유하고 수직칸에 2개 단추를 보관한다.

목록칸은 43행에서 만든다. 43행과 44행의 순환은 목록칸에 유표이름들을 추가한다. 46

행에서 connect()호출은 changeCursor()라는 이름의 국부처리부와 highlighted()라는 목록칸신호를 련결한다. changeCursor()는 목록칸성원을 선택할 때마다 실행된다. 여기서 그것을 사용하지 않지만 목록칸성원우에서 두번찰칵을 요구하는 목록칸으로부터 오는 selected()신호는 있다. 두 목록칸신호들은 목록칸항목의 첨수번호를 제공한다.

selectButton은 51행에서 만들어지고 국부처리부 selectCursor()에 련결된 clicked()신호를 가지고있다. exitButton은 57행에서 만들어지며 국부처리부 shutdown()에 련결된 clicked()를 가지고있다.

68행의 처리부메쏘드 changeCursor()에는 매번 최근에 선택한 목록칸항목의 첨수가 넘어온다. 목록칸의 본문은 배렬이름으로부터 적재되였으므로 목록칸으로부터 오는 첨수도 역시 배렬의 첨수이다. 70행은 배렬로부터 유표 ID를 꺼내서 cursorID에 보관하고 그것을 현재 유표ID번호로 한다. 71행에서 setCursor()호출은 인수로서 유표를 제공된것으로 설정한다. QCursor구성자는 목록칸으로부터 선택될 때 미리 정의된 유표들중 하나의 유표 ID를 받아들인다.

73행에서 처리부메쏘드 selectCursor()는 selectButton의 유표를 현재 선택한 유표로 설정한다. 이 단추에 어떤 유표도 설정되지 않았다면 그것은 그 부모창문(즉 71행에서 정의한 유표)과 같은것을 사용한다. 75행에서 일단 setCursor()호출이 이루어지면 selectButton은 그 자체의 유표를 사용한다.

이 모든것은 3개 유표가 하나의 응용프로그람에 사용된다는것을 의미한다. 기정유표(ArrowCursor)는 기본창문의 제목띠에 계속 사용된다. 현재 목록칸에서 선정되는 어느 유표나 기본창문내부의 어디서나 사용된다. 유일한 례외는 Select단추가 그것을 눌렀을 때 선정된 자기의 유표를 가지고있는것이다.

목록에 2개의 특수유표가 있다. BlankCursor라는 유표는 도형을 가지지 않고 유표는 표시되지 않는다. 그것은 여전히 이동하여 찰칵할수 있는 유표이지만 그것을 알수 없다. 다른 특수한 유표는 BitmapCursor이다. 이것이 동작하자면 자기가 만든 유표를 제공하여야 하는데 그것은 다음 절에서 보게 된다.

제5절. 자체의 유표설계

유표는 화소들로 이루어진 직4각형이다. 매개 화소는 흑색, 백색이거나 투명색일수 있다. 실례로 KDE기정화살유표는 흰 테두리를 가지는 흑색화살표이다. 다른 모든 화소들은 투명색이다. 한가지 색으로 칠하고 다른 색으로 테두리를 그리면 유표가 지나는 배경색에 관계없이 유표를 볼수 있다.

유표를 만들려면 2개의 비트매프를 만들어야 한다. 두 비트매프는 그들중 하나가 다른것과 겹치는 마스크이므로 같은 높이와 너비를 가져야 한다. 화소당 1bit만 있으므로 마스크가 필요하지만 화소를 현시하는 방법은 3가지 있다. 즉 흑색, 백색, 혹은 투명색.

그림 8-4에서 16×16유표형태를 보여준다. 제일 우의 화소는 그 안에 다이야몬드가 있

으며 이 유표의 열점(hot spot)으로 된다. 열점은 보통 지시기의 끝 또는 +자의 중심에 있으며 응용프로그람에 통보되는 정확한 유표위치를 결정한다. 그림 8-5에서는 유표용마스크로서 동작하는 다른 비트매프를 보여준다. 두 비트매프는 표 8-1의 규칙에 따라 결합된다.

표 8-1. 유표비트매프현시규칙

유표비트설정	유표마스크비트설정	결과
1	1	흑색
0	1	백색
0	0	투명
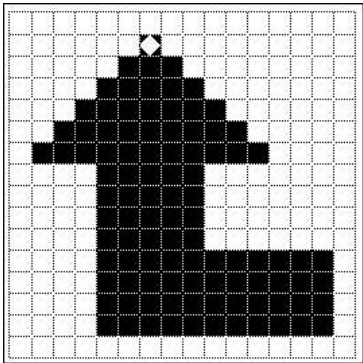
그림 8-4. 유표형태의 정의
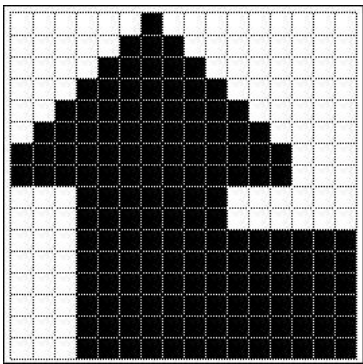
그림 8-5. 유표마스크의 정의

유표를 설명하는 비트매프를 만드는데 쓰이는 몇가지 편의프로그람들이 있다. 하나는 X11체계의 부분으로 제공된다. 이것은 보통 /usr/X11R6/bin/bitmap에 있고 16×16유표화상을 만드는데 사용한다. 프로그람실행을 시작하고 마우스로 화소들을 선택한다. 일반적으로 이 편의프로그람과 비트매프에 대해서는 9장에서 설명한다.

다음 실례는 유표를 만들기 위하여 그림 8-4와 8-5에 보여준 유표비트매프를 사용한다.

MyCursor머리부파일

```
1 /* mycursor.h */
2 #ifndef GRABMOUSE_H
3 #define GRABMOUSE_H
```

```
4
5 #include <qwidget.h>
6
7 class MyCursor: public QWidget
8 {
9 public:
10     MyCursor(QWidget *parent=0,const char *name=0);
11 };
12
13 #endif
```

MyCursor

```
1 /* mycursor.cpp */
2 #include <kapplication.h>
3 #include <qcursor.h>
4 #include <qbitmap.h>
5 #include "mycursor.h"
6
7 #define upleft_width 16
8 #define upleft_height 16
9 #define upleft_x_hot 6
10 #define upleft_y_hot 1
11 static unsigned char upleft_bits[] = {
12     0x00, 0x00, 0x40, 0x00, 0xe0, 0x00, 0xf0, 0x01, 0xf8,
13     0x03, 0xfc, 0x07, 0xfe, 0x0f, 0xf0, 0x01, 0xf0, 0x01,
14     0xf0, 0x01, 0xf0, 0x01, 0xf0, 0x7f, 0xf0, 0x7f, 0xf0,
15     0x7f, 0xf0, 0x7f, 0x00, 0x00};
16
17 #define upleftmask_width 16
18 #define upleftmask_height 16
19 static unsigned char upleftmask_bits[] = {
20     0x40, 0x00, 0xe0, 0x00, 0xf0, 0x01, 0xf8, 0x03, 0xfc,
21     0x07, 0xfe, 0x0f, 0xff, 0x1f, 0xff, 0x1f, 0xf8, 0x03,
22     0xf8, 0x03, 0xf8, 0xff, 0xf8, 0xff, 0xf8, 0xff, 0xf8,
23     0xff, 0xf8, 0xff, 0xf8, 0xff};
```

```
24
25 int main(int argc, char **argv)
26 {
27    KApplication app(argc, argv, "mycursor");
28    MyCursor *mycursor = new MyCursor();
29    mycursor->show();
30    app.setMainWidget(mycursor);
31    return(app.exec());
32 }
33
34 MyCursor::MyCursor(QWidget *parent,const char *name)
35    : QWidget(parent,name)
36 {
37    QBitmap upleft(upleft_width,upleft_height,
38            upleft_bits,TRUE);
39    QBitmap upleftmask(upleftmask_width,upleftmask_height,
40            upleftmask_bits,TRUE);
41    QCursor upleftCursor(upleft,upleftmask,
42            upleft_x_hot,upleft_y_hot);
43    setCursor(upleftCursor);
44    resize(100,100);
45 }
```

유표본체는 7～15행에서 정의된다. 이 형식의 자료는 비트매프편의프로그람으로부터의 실제 출력이며 간단히 프로그람의 원천코드에 삽입되였다. 유표의 높이와 너비, 그리고 열점의 위치는 모두 상수로 정의된다. 자기가 좋아하는 크기의 유표를 만들수 있으나 대부분의 유표는 16×16 또는 32×32화소들이다. 유표마스크에 사용하는 비트매프는 17～23행에서 정의된다.

비트매프자료를 유표로 변환하는 수속은 34행에서 시작하는 구성자에 있다. 37행에서 QBitmap객체를 만드는데 유표비트매프자료를 사용한다. 같은 방법으로 39행에서 QBitmap을 만들 때 마스크를 사용한다. 정의된 상수들은 QBitmap구성자에서 비트배렬에 적용할 높이와 너비를 결정하는데 필요하다. 끝으로 두 비트매프와 열점의 위치를 사용하여 41행에서 유표를 만든다. 43행의 setCursor()호출은 현재창문에 유표화상을 설치한다. 결과는 그림 8-6에 보여준다.
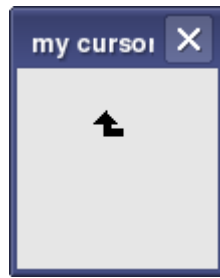
그림 8-6. 사용자정의유표

제6절. 건반사건

건반은 보통 2개 사건을 발생시킨다. 하나는 건을 누를 때, 다른 하나는 건을 놓을 때 발생한다. 이것은 쏘프트웨어가 어떤 건결합을 누르고있는가 결정할수 있게 한다. 실례로 Shift건을 누르고 놓지 않는다면 문자건은 소문자가 아니라 대문자로 인식된다. Qt서고는 건들을 정의한다.

다음 프로그람은 그림 8-7에 보여준 2구획창문을 현시한다. 오른쪽의 구획은 건누르기 정보를 받아들이도록 설정되고 매개 건치기는 왼쪽구획에 목록된다. 매개 행은 Press 혹은 Release사건인가를 가리키는 P 또는 R로 시작한다. 두점뒤의 수는 건의 유일식별번호이다. 그뒤에 건서술이 온다(그러나 이 실례에서는 설명이 없다). 건반사건이 건을 누르고있기때문에 생성되였다면 단어 "repeat"가 오른쪽에 나타난다.
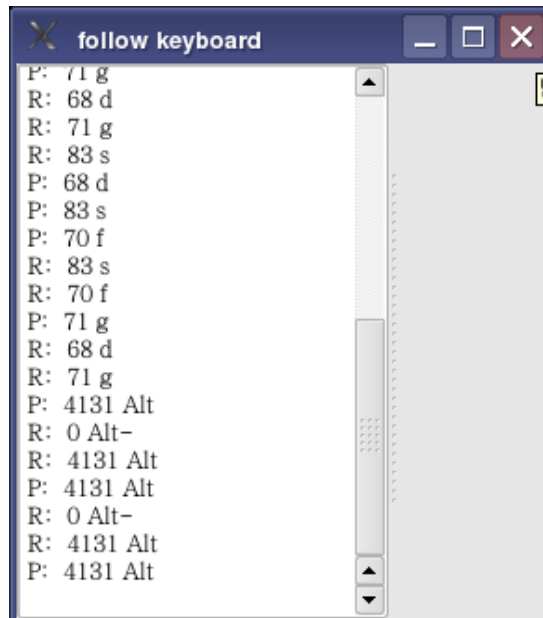
그림 8-7. 매개 건치기로부터 오는 정보의 현시

> 알아두기: Qt는 약 240개의 유일건을 인식한다. 그러나 Control, Alt 및 Shift건결합과 특수건들과 특수문자를 가지고있으므로 모두 약 2000개나 된다. 특수건에 대한 전체목록은 파일 qnamespace.h에 있다.

FollowKeyboard머리부파일

```
1 /* followkeyboard.h */
2 #ifndef FOLLOWKEYBOARD_H
3 #define FOLLOWKEYBOARD_H
4
5 #include <qsplitter.h>
6 #include <qstring.h>
7 #include <qmultilineedit.h>
8
9 class FollowKeyboard: public QSplitter
10 {
11     Q_OBJECT
12 public:
13     FollowKeyboard(QWidget *parent=0,const char *name=0);
14 public slots:
15     void newline(QString &);
16 private:
17     QMultiLineEdit *edit;
18 };
19
20 #endif
```

FollowKeyboard

```
1 /* followkeyboard.cpp */
2 #include <kapplication.h>
3 #include <qstring.h>
4 #include "keyboardsensor.h"
5 #include "followkeyboard.h"
6
7 int main(int argc, char **argv)
8 {
9     KApplication app(argc, argv, "followkeyboard");
10    FollowKeyboard *followkeyboard = new FollowKeyboard();
11    followkeyboard->show();
12    app.setMainWidget(followkeyboard);
```

```
13    return(app.exec());
14 }
15
16 FollowKeyboard::FollowKeyboard(QWidget *parent,
17    const char *name) : QSplitter(parent,name)
18 {
19    edit = new QMultiLineEdit(this);
20    edit->setMinimumWidth(80);
21    edit->setReadOnly(TRUE);
22    edit->setMinimumWidth(200);
23
24    KeyboardSensor *sensor = new KeyboardSensor(this);
25    sensor->setMinimumWidth(80);
26
27    connect(sensor,SIGNAL(description(QString &)),
28            this,SLOT(newline(QString &)));
29
30    resize(10,10);
31 }
32 void FollowKeyboard::newline(QString &str)
33 {
34    edit->insertLine(str);
35    edit->setCursorPosition(5000,0);
36 }
```

16행에서 시작하는 FollowKeyboard구성자는 2개 창문부품을 가지고있는 수평QSplitter창문부품이다. 왼쪽에 있는것은 본문을 현시하는데 사용되는 QMultiLineEdit객체이고 오른쪽에 있는것은 건치기정보를 받아들이는데 사용되는 KeyboardSensor창문부품이다. 32행의 처리부메쏘드 newline()은 QMultiLineEdit객체에서 본문의 아래에 문자렬을 추가한다. 35행의 setCursorPosition()호출은 본문창문의 제일 끝행을 표시한다.

KeyboardSensor머리부파일

```
1 /* keyboardsensor.h */
2 #ifndef KEYBOARDSENSOR_H
3 #define KEYBOARDSENSOR_H
4
5 #include <qwidget.h>
```

```
6 #include <qevent.h>
7 #include <qstring.h>
8
9 class KeyboardSensor: public QWidget
10 {
11    Q_OBJECT
12 public:
13    KeyboardSensor(QWidget *parent=0,const char *name=0);
14 private:
15    void emitDescription(const QString &,QKeyEvent *);
16    virtual void keyPressEvent(QKeyEvent *event);
17    virtual void keyReleaseEvent(QKeyEvent *event);
18 signals:
19    void description(QString &);
20 };
21
22 #endif
```

이 머리부파일은 클라스 KeyboardSensor를 QWidget의 파생클라스로 선언한다. 16행과 17행에서 keyPressEvent()와 keyReleaseEvent()의 선언은 QWidget기초클라스의 함수들을 재정의하며 그것들은 매개 건치기에 대하여 한번 호출된다. 19행의 신호 description()은 매개 건치기의 설명을 생성한다.

KeyboardSensor

```
1 /* keyboardsensor.cpp */
2 #include <qstring.h>
3 #include <ctype.h>
4 #include "keyboardsensor.h"
5
6 KeyboardSensor::KeyboardSensor(QWidget *parent,
7      const char *name) : QWidget(parent,name)
8 {
9      setFocusPolicy(QWidget::StrongFocus);
10     setMinimumSize(300,300);
11 }
12 void KeyboardSensor::keyPressEvent(QKeyEvent *event)
13 {
```

```
14    emitDescription(QString("P: "),event);
15 }
16 void KeyboardSensor::keyReleaseEvent(QKeyEvent *event)
17 {
18    emitDescription(QString("R: "),event);
19 }
20 void KeyboardSensor::emitDescription(
21    const QString &typeStr,QKeyEvent *event)
22 {
23    int key = event->key();
24    int ascii = event->ascii();
25    ButtonState state = event->state();
26
27    QString keyStr = QString("%1").arg(key);
28
29    QString charStr = QString("");
30    if(key == Key_Control) {
31            charStr += QString("Control");
32    } else if(key == Key_Alt) {
33            charStr += QString("Alt");
34    } else if(key == Key_Shift) {
35            charStr += QString("Shift");
36    } else {
37            if(state & ControlButton)
38                    charStr += "Ctl-";
39            if(state & AltButton)
40                    charStr += "Alt-";
41            if(state & ShiftButton)
42                    charStr += "Shft-";
43            if(isgraph(ascii))
44                    charStr += ascii + QString(" ");
45            else if(state & ControlButton)
46                    charStr += (ascii + 64) + QString(" ");
47    }
48
```

```
49    if(event->isAutoRepeat())
50            charStr += " repeat";
51
52    QString str = QString("%1 %2 %3")
53            .arg(typeStr).arg(keyStr).arg(charStr);
54
55    emit description(str);
56 }
```

임의의 시각에 현시기우의 1개 창문만 건반초점을 가지며 입력된 건치기는 그 1개 창문으로만 간다. 일부 창문은 초점을 받아들일수 있고 다른 창문들은 그렇지 못하다. 6행의 구성자는 이 창문부품이 건반초점을 받아들이게 하는 방법을 지정하기 위해 setFocusPolicy()를 호출한다. 기정으로 창문부품은 건반초점을 받아들이지 않는다. 표 8-2는 가능한 초점환경설정을 설명한다.

표 8-2. 초점방식을 조종하는 설정

이름	설명
ClickFocus	마우스에 의해 선택되기만 하면 초점은 이 창문부품으로 옮긴다.
NoFocus	이 창문부품은 초점을 받아들이지 않는다.
StrongFocus	이 방식은 TabFocus와 ClickFocus의 결합이다.
TabFocus	오직 Tab건만이 한 창문부품에서 다른 창문부품으로 초점을 옮기는데 사용될수 있다. 이것은 보통 대화칸에서 창문부품집합에 사용된다.
WheelFocus	초점은 TabFocus, ClickFocus, 또는 마우스바퀴의 이동을 사용하여 이 창문부품으로 변경될수 있다.

12～19행에서 메쏘드 keyPressEvent()와 keyReleaseEvent()는 기초클라스의 가상메쏘드들을 재정의하며 매개 건치기와 함께 호출된다. 두 메쏘드는 QKeyEvent객체를 넘겨받고 메쏘드 emitDescription( )에 넘기여 문자렬로 형식화한다.

20행에서 시작하는 메쏘드 emitDescription()은 QKeyEvent로부터 자료를 꺼내서 설명문자렬을 가진 description()신호를 발생시킨다.

매개 건은 유일번호를 가지고있다. 유일번호는 27행에서 key()호출에 의해 QkeyEvent로부터 얻어진다. 건이 현시가능한 ASCII문자라면 건값은 그 ASCII값과 같다. 현시할수 없는 건에는 더 큰 수가 할당된다. 실례로 그림 8-7에서 이미 본것처럼 Shift건의 값은 4128, Alt건은 4131 그리고 Return건은 4100이다.

조건코드(30～35행)는 건이 3개 수식건중의 하나인가를 판단한다. 그것이 아니면 37～46행의 코드는 누르고있는 수식자들을 렬거하고 그다음 문자자체(현시할수 있다면)를 현시한다. 문자를 현시할수 있다면 43행의 시험은 참이다. 그러나 Control건을 누르고있으면 문자값자체는 수식된다(어떠한 Control문자에 64를 더하면 원래의 현시가능한 문자가 나타난다).

49행은 Boolean메쏘드 isAutoRepeat()를 검사하여 건을 누르고있는가 판단하고 사건들을 자동적으로 생성한다.

요 약

Qt서고는 마우스와 건반으로부터 오는 자료의 처리를 간단화한다. 사건들은 서술객체로 형식화되고 정확한 창문부품으로 발송된다. 이 장은 다음과 같은 내용을 설명하였다.

- 하드웨어에서의 사건과 응용프로그람에서의 메쏘드호출사이에는 직접관계가 있을수 있다.
- 어떤 창문부품이나 이동과 마우스단추찰칵 모두에 대하여 마우스를 감시할수 있다.
- 마우스유표의 형태변경은 자식창문이 그 자체의 변화를 일으키지 않으면 그 부모창문과 자식창문들 모두에 대해서만 형태를 변경한다.
- 건사건들은 모든 건누르기와 모든 건놓기때에 발생된다.

이 장은 사용자유표를 만드는데 요구되는 도형처리에 대하여 간단히 언급하였다.

제9장. 도형파일형식

학습내용

2가지 기본형식의 도형자료의 조작

파일에 보관된 도형자료의 적재와 현시

그림기호들을 가지는 단추와 표식자들의 만들기

각이한 상태들을 표시하기 위하여 특수한 종류의 도형

이 장은 주로 디스크파일들로부터 도형자료를 적재하여 현시하는 방법을 설명한다. KDE는 많은 형식의 도형파일을 인식하고 읽을수 있다. 현시기에 나타나는 모든것은 QWidget로부터 계승되므로 하나의 창문을 가진다. 또한 창문을 가지는 모든 클라스는 창문안에 픽스매프(색도형)를 현시할수 있다.

프로그람은 두 위치중 하나로부터 그 도형자료를 얻을수 있다. 그것은 디스크의 파일에 여러 형식들중 하나로 보관되고 프로그람은 그다음 파일을 읽어들이고 자료를 내부픽스매프로 변경한다. 또한 도형파일의 내용을 자기 프로그람으로 직접 콤파일할수 있도록 하기 위하여 C원천코드로 변환하는 방법이 있다. 2가지 방법은 같은것 즉 창문을 도색하는데 쓰이는 QPixmap객체에 의거한다.

제1절. 두 종류의 도형

2가지 기본종류의 도형은 비트매프와 픽스매프이다.

• 픽스매프는 화소값들의 직4각형배렬이다. 배렬에서 매개 값은 1개 화소의 색을 표시한다. 픽스매프는 임의의 시간에 자기 조색판에 적재할 때와 같이 많은 색을 담을수 있다.

• 비트매프는 1개 화소에 매개 비트가 대응하는 비트들의 직4각형배렬이다. 비트매프는 2가지 색만 가지고있다. 즉 매개 화소는 on 또는 off이다. 보통 이것은 흑백색으로 현시되지만 KDE는 임의의 두 색을 사용하여 비트매프를 현시할수 있다. 비트매프는 실제로 픽스매프의 특수한 경우이지만 그것은 흔히 그자체의 특수한 파일형식으로 사용된다.

도형파일형식은 끝없이 많다. universal변환프로그람에 의거하면 임의의 도형파일형식이라도 KDE응용프로그람에서 사용할수 있다. convert편의프로그람(이 장에서 후에 자세히 설명)은 도형파일을 외부형식으로부터 현시가능한 형식으로 변환할수 있다. 실례로 다음 지령은 JPEG파일을 프로그람으로 직접 콤파일할수 있는 형식인 픽스매프로 변환하는 방법을 보여준다.

```
convert rickrack.jpeg rickrack.xpm
```

프로그람에 비트매프(흑백색)를 포함하려면 다음과 같이 변환할수 있다.

```
convert rickrack.jpeg rickrack.xbm
```

convert편의프로그람은 입력파일의 내용을 보고 그것이 어떤 종류의 파일인가를 결정하고(그것은 입력에서 파일확장자에 의거하지 않는다) 출력파일이름의 확장자를 보고 어떤 종류의 도형파일을 생성하는가 결정한다.

제2절. XPM형식

XPM(XPixMap)도형형식은 ASCII본문으로 도형을 보관하기 위한 X11표준이다. 이 형식은 문서편집기를 사용하여 간단한 색도형을 만들거나 변경할수 있게 한다. XPM정의는 ASCII일뿐아니라 그 형식은 직접 자기 프로그람으로 콤파일할수 있는 C원천코드이다.

다음은 4색을 가지는 XPM도형의 실례이다.

```
1 /* XPM */
2 /** essPixmap.xpm **/
3 static const char *essPixmap[] = {
4    "12 14 4 1",
5    "   c None",
6    "X c #FFFFFF",
7    "R c Red",
8    "B c #0000FF",
9    "      RRBB      ",
10   "XXXXXXXXXXXX",
11   "XXXXXXXXXXXX",
12   "XX   RRBB      ",
13   "XX   RRBB      ",
14   "XX   RRBB      ",
15   "XXXXXXXXXXXX",
16   "XXXXXXXXXXXX",
17   "      RRBB   XX",
18   "      RRBB   XX",
19   "      RRBB   XX",
20   "XXXXXXXXXXXX",
21   "XXXXXXXXXXXX",
22   "      RRBB      ",
23 };
```

XPM파일의 문법은 문자렬들의 배렬로서 정의된다. 첫 행의 설명문은 편의프로그람들이 파일형을 결정하는데 그것을 사용하므로 반드시 있어야 한다.

4행은 뒤에 오는 자료를 설명하는데 사용되는 4개 수를 포함한다. 첫째 수는 픽스매프가 12화소너비라는것을 서술하며 둘째 수는 그것이 14화소 높이라는것을 서술한다. 다음 수는 4개 색이 도형을 그리는데 사용된다는것을 지정한다.

마지막 수자는 한개 문자를 매개 색의 표식자로 사용한다는것을 지정한다.

5∼9행은 색정의이다. 매개 문자렬은 색을 식별하는 표식자로 쓰이는 문자로 시작한다.

어떠한 ASCII문자든 사용할수 있다. 5행은 None이라는 이름으로 공백문자를 정의한다. 이것은 어떤 화소도 색칠하지 않는다는것을 지정하며 배경을 다시 쓰지 않으므로 투명색을 생성한다. 6행은 문자 X에 흰색의 값을 지정한다. 16진수값 FFFFFF는 흰색용의 적록청값이다(10진수일 때 값들은 255 255 255이다). 8행은 문자 B에 청색을 지정하는데 16진수값 0000FF를 사용한다. 7행은 문자 R의 색을 정의하는데 어떤 이름을 사용한다. 그 이름은 파일 /usr/X11R6/lib/X11/rgb.txt에 있는 RGB이름들중의 하나이여야 한다.

도형자체는 9행에서 시작하고 22행에서 끝난다. 도형의 너비가 12화소이므로 매개 문자렬의 너비는 12화소이며 화소를 표시하는데 오직 1문자가 사용된다. 도형의 높이가 14화소이므로 이런 문자렬이 14개 있다. 매개 화소는 초기에 정의된 4색문자들중 하나를 포함하여 값을 할당한다.

XPM파일은 많은 색을 가지는 큰 고분해능화상들을 표시하는데 사용될수 있다. 실례로 Linux배포물에는 Linux펭긴새를 포함하는 logo.gif라는 파일이 있다. 다음 지령으로 GIF파일을 XPM파일로 바꿀수 있다.

```
convert logo.gif logo.xpm
```

그때 생기는 XPM파일에서 24bit이상의 색정보가 포함되는데 이것은 단일문자로 표시할수 있는것이상 많은 색이 있다는것을 의미한다. 전체 XPM파일은 560행이다. 여기에서 발취한것이 있다.

```
/* XPM */
static const char *magick[] = {
   "257 303 251 2",
   " c Gray0",
   ". c #080800000404",
   "X c #080808080000",
   "o c Gray3",
   "O c #101004040404",
   "+ c #101010100404",
   . . .
   "{. c #f0f0b8b80808",
   "}. c #f8f8b0b00808",
   "|. c #f8f8b8b80808",
   " X c #f0f0b0b01010",
   ".X c #f0f0b8b81010",
   "XX c #f8f8b8b81010",
   "BX c #d8d8d8d8e8e8",
```

```
    "VX c #e0e0e0e0d8d8",
    "CX c #f0f0e8e8d8d8",
    "ZX c Gray88",
    "AX c Gray91",
    "SX c #e8e8e8e8f0f0",
    "DX c #f0f0e8e8ecec",
    "FX c #f0f0f0f0e8e8",
    "GX c Gray94",
    "HX c #f8f8f8f8f8f8",
    "JX c None",
    . . .
```

이 XPM도형은 257×303화소크기이다. 그것은 모두 251개 색을 포함하며 매개 색을 표시하는데 2문자를 리용한다. 처음 몇개 문자들은 단일문자에 의해 정의되는것 같지만 사실상 공백이 둘째 문자로 되므로 2문자 리용한다. 파일을 설명하는 부분에서 볼수 있는것처럼 종지부와 X문자가 사용된다. 색을 정의하는데 2문자 요구되므로 화소값의 행을 정의하는 매개 문자렬은 514개 문자(257의 두배)를 가져야 한다.

또한 색들의 16진수는 6개가 아니라 12개 수자를 가지고있다. 이것은 여전히 RGB형식이지만 매개 색은 16bit(4개 16진수)이다. 어느쪽이든 길이는 XPM파일에 유효하다. 그것을 읽어들이는 쏘프트웨어는 형식을 결정하기 위한 수자들을 계수한다. 파일 /usr/X11R6/lib/X11/rgb.txt안의 색들과 다른 곳에서 찾은 많은 색들은 3개의 8bit값으로 정의된다. 다음의 간단한 프로그람은 3개의 8bit값들을 XPM에서 요구되는 long과 short 16진문자렬들로 변환한다.

```
/* hexcolor */
#include <stdio.h>
#include <stdlib.h>
char *usage[] = {
    " Usage: hexcolor r g b",
    " Enter the three RBG color values in the",
    " range of 0 to 256. The output is both a",
    " 24-bit and 48-bit hexadecimal number of the",
    " color that can be used in an XPM file."
};
int main(int argc,char *argv[])
{
    int i;
```

```
    int r,g,b;
    if(argc < 4) {
        for(i=0; i<5; i++)
            printf("%s\n",usage[i]);
        exit(1);
    }
    r = atoi(argv[1]);
    g = atoi(argv[2]);
    b = atoi(argv[3]);
    printf("#%02X%02X%02X\n",r,g,b);
    printf("#%02X00%02X00%02X00\n",r,g,b);
    exit(0);
}
```

제3절. 자료로부터 XPM의 표시

사실상 convert편의프로그람은 임의의 도형파일을 XPM파일로 변환할수 있고 XPM형식이 C 원천코드이므로 어떠한 도형이라도 자기 프로그람으로 직접 콤파일할수 있다. 이것은 대체로 그림기호, 단추표식자, 목록항목 그리고 다른 작은 장식항목들에 사용된다.

> 알아두기: convert편의프로그람은 C++가 아니라 C로 화상자료를 변환한다. 실제로 프로그람내에서 QPixmap를 만들려면 XPM파일을 사용하기전에 제일 웃행을 편집하여 이름선언에 const변경자를 삽입해야 한다. const없이도 제대로 콤파일되지만 QPixmap구성자는 그것이 const가 아니라고 통보한다. 그리고 프로그람에서 하나이상의 XPM파일을 사용하려고 한다면 항상 convert가 그것을 magick라고 이름을 붙이므로 배렬에 새로 이름을 붙여야 한다.

다음의 프로그람은 XPM파일을 직접 콤파일하여 현시하는 실례이다.

```
1 /* showxpm.cpp */
2 #include <kapplication.h>
3 #include <qwidget.h>
4 #include <qpixmap.h>
5
6 #include "logo.xpm"
7
8 int main(int argc, char **argv)
9 {
10    KApplication app(argc, argv,  "showxpm");
```

```
11    QPixmap pixmap(magick);
12    QWidget *widget = new QWidget();
13    widget->setFixedSize(pixmap.width(),pixmap.height());
14    widget->setBackgroundPixmap(pixmap);
15    widget->show();
16    app.setMainWidget(widget);
17    return (app.exec());
18 }
```

6행의 #include문은 XPM자료가 프로그람으로 직접 콤파일되게 한다. 11행은 XPM자료로부터 QPixmap를 만든다.

어떠한 창문부품라도 그 배경색으로서 픽스매프를 현시하는데 사용될수 있으므로 이 실례에서 일반창문부품은 12행에서 만들어진다. 13행의 setFixedSize()호출은 창문부품이 픽스매프와 꼭같은 크기로 되게 한다. 14행의 setBackgroundPixmap()호출은 창문부품에 픽스매프를 삽입한다. 결과는 그림 9-1에 보여준다.
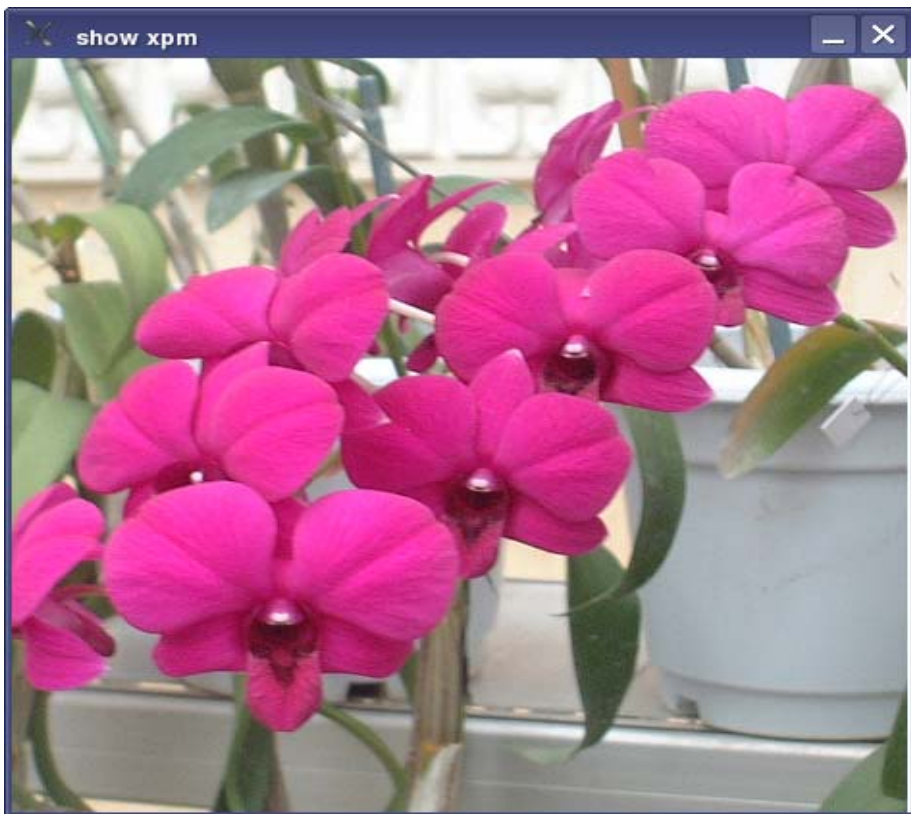
그림 9-1. 콤파일된 XPM자료의 현시

고정크기로 창문을 설정하고 정확한 크기의 픽스매프를 가지고있다면 전체 픽스매프가 표시된다. 창문부품의 창문이 픽스매프보다 작다면 화상은 아래로, 오른쪽으로 잘리운다. 창문이 픽스매프보다 크다면 픽스매프는 창문이 가득 찰 때까지 타일형태로 붙여진다. 다음

실례는 더 작은 픽스매프(이 장에서 초기에 essPixmap라고 이름지었다)를 사용하여 창문부품 배경에 타일을 붙인다.

```
1 /* showxpmtile.cpp */
2 #include <kapplication.h>
3 #include <qwidget.h>
4 #include <qpixmap.h>
5
6 #include "essPixmap.xpm"
7
8 int main(int argc, char **argv)
9 {
10     KApplication app(argc, argv,  "showxpmtile");
11     QPixmap pixmap(essPixmap);
12     QWidget *widget = new QWidget();
13     widget->setBackgroundPixmap(pixmap);
14     widget->resize(200,100);
15     widget->show();
16     app.setMainWidget(widget);
17     return (app.exec());
18 }
```

이 실례는 6행에서 원천에 포함된 XPM자료로부터 작은 픽스매프를 만든다. 픽스매프의 실제크기를 무시하고 그것이 13행에서 배경픽스매프로서 설정되며 창문부품의 크기는 14행에서 설정된다. 결과는 그림 9-2에 보여주는 창문이다.
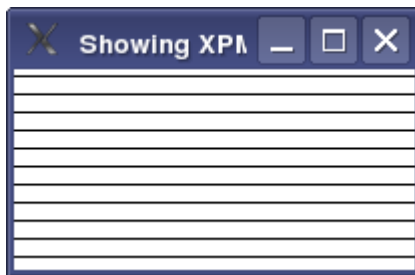
그림 9-2. 콤파일된 XPM자료를 타일배경으로 현시

제4절. 파일로부터 픽스매프의 적재

이전의 실례를 약간 변경함으로써 도형을 프로그람의 부분으로서 콤파일하지 않고 파일로부터 실을수 있다. 필요한것은 다른 방법으로 픽스매프를 만드는것이다. 다음의 프로그람은 이미 그림 9-1에서 보여준 logo픽스매프를 적재하고 현시한다.

```
1 /* showfilexpm.cpp */
2 #include <kapplication.h>
3 #include <qwidget.h>
4 #include <qpixmap.h>
5
6 int main(int argc, char **argv)
7 {
8     KApplication app(argc, argv,  "showfilexpm");
9     QPixmap pixmap("logo.xpm");
10    QWidget *widget = new QWidget();
11    widget->setFixedSize(pixmap.width(),pixmap.height());
12    widget->setBackgroundPixmap(pixmap);
13    widget->show();
14    app.setMainWidget(widget);
15    return  (app.exec());
16 }
```

9행에서 QPixmap구성자는 도형자료의 위치를 파일을 사용하여 지정한다. XPM이외의 파일형들을 사용할수 있다. 실례로 각이한 형의 파일을 싣기 위하여 다음과 같이 9행에서 파일이름을 변경한다.

```
QPixmap pixmap(“logo.gif”);
```

쏘프트웨어는 파일이름의 뒤붙이를 보지 않고 파일형을 결정한다. 그대신에 파일선두의 자료블로크를 싣고 그것을 검사하여 파일형을 결정한다. 이것은 문자렬 XPM을 포함하는 설명문이 달린 문자렬이 XPM파일의 선두에 있어야 하는 리유이다.

쏘프트웨어가 무효한 XPM파일이라고 통보한다면 그것은 낡은 형식일수 있다. 파일을 사용하려면 판번호가 3이여야 한다. 다음과 같은 지령으로 변환한다.

```
sxpm -nod oldform.xpm -o newform.xpm
```

도형파일을 적재하려면 쏘프트웨어는 파일형식을 리해하여야 한다. Qt쏘프트웨어는 파일형으로서 PNG, BMP, GIF, JPEG, XBM, XPM 그리고 PNM을 제공한다. Qt쏘프트웨어는 후에 다른 형식을 포함하도록 확장할수 있게 설계되였다.

여러가지 파일형식들로부터 읽어들이는데 특별히 요구되는것은 없다. 다음의 실례프로

그람은 그림 9-1에 이미 보여주는 JPEG판의 도형을 적재하고 현시한다.

```
/* showfilejpeg.cpp */
#include <kapplication.h>
#include <qwidget.h>
#include <qpixmap.h>
int main(int argc, char **argv)
{
    KApplication app(argc, argv,  "showfilejpeg");
    QPixmap pixmap("logo.jpeg");
    QWidget *widget = new QWidget();
    widget->setFixedSize(pixmap.width(),pixmap.height());
    widget->setBackgroundPixmap(pixmap);
    widget->show();
    app.setMainWidget(widget);
    return(app.exec());
}
```

제5절. 픽스매프를 리용한 단추의 장식

단추는 다른 창문부품처럼 창문을 포함하므로 그것은 본문뿐아니라 그림을 현시할수 있다. 사실상 QPushButton클라스는 픽스매프가 단추의 현재상태를 표시하도록 하는 몇가지 특별한 확장기능을 가지고있다. 다음 프로그람은 PNG파일을 사용하여 그림 9-3에 보여준 단추의 앞면을 칠한다.

```
1 /* decobutton.cpp */
2 #include <kapplication.h>
3 #include <kpixmap.h>
4 #include "decobutton.h"
5
6 int main(int argc, char **argv)
7 {
8      KApplication app(argc, argv,  "decobutton");
9      DecoButton decobutton;
10     decobutton.show();
11     app.setMainWidget(&decobutton);
12     return  (app.exec());
13 }
```

```
14
15 DecoButton::DecoButton(QWidget *parent,const char *name)
16    : QWidget(parent,name)
17 {
18    setFixedSize(200,150);
19
20    QPixmap pixmap("hil-app-go.png");
21    button = new QPushButton(this);
22    button->setPixmap(pixmap);
23    button->setGeometry(50,50,100,50);
24 }
```
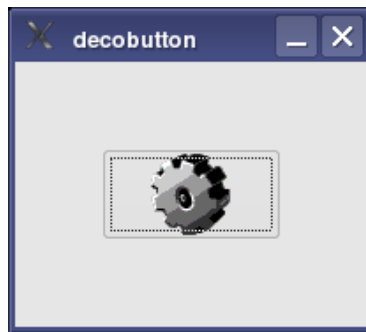
그림 9-3. 본문대신 도형을 가지는 단추

제일웃준위창문에 사용한 창문부품은 DecoButton이다. 구성자는 15행에서 시작한다. 18행은 200×150화소의 고정크기로 창문부품을 설정한다.

픽스매프는 20행에서 hil-app-go.png라는 파일로부터 만들어진다. 단추가 만들어지고 픽스매프는 22행에서 setPixmap()호출에 의해 안에 삽입된다. 23행에서 단추는 이 픽스매프에 알맞는 크기가 설정된다.

QPushButton클라스는 단추를 누를 때 특별한 일을 한다. 단추가 눌리운것으로 보이도록 하기 위하여 배경은 더 어두운 색으로 바뀌고 도형 그자체는 1화소 아래로, 1화소 오른쪽으로 옮겨진다. 결과는 그림 9-4에 보여준다.
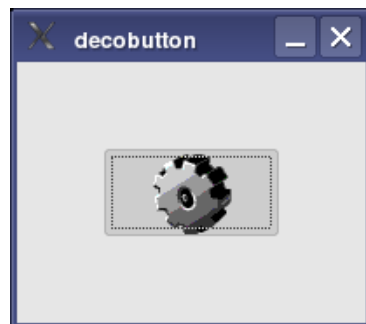
그림 9-4. 본문대신에 도형으로 능동화된 단추

단추가 능동으로 될 때 그림기호에 의해 가리우지 않은 구역은 흐려진다. 그림기호는 실제로 바른4각형이므로 도형자체안의 일부 화소들을 흐리게 하기 위하여 도형을 투명색으로 하여야 한다. 단추를 선택할 때 색을 변경하는 투명색화소들은 사용자에게 좋은 효과를 준다. 그러나 그림기호 자체의 도형이 수정되였다는것은 그림 9-3과 그림 9-4를 비교하여야만 알수 있다. 이 장에서 후에 서술하는 QIconSet클라스에서 다른 방법을 고찰한다.

제6절. XBM형식

2개 색(보통 흑색과 백색)만 있으면 XBM(XBitMap)형식으로 매개 화소에 1bit를 가진 그림을 보관하는것이 더 효과적이다. XBM형식은 보통 마우스와 건반유표를 정의하는데 사용되지만 또한 다른 목적도 가지고있다. XPM형식처럼 XBM파일은 C프로그람으로 직접 콤파일할수 있는 ASCII파일이다. 다음은 XBM파일형식의 실례이다.

```
#define arrow_width 16
#define arrow_height 16
#define arrow_x_hot 15
#define arrow_y_hot 7
static unsigned char arrow_bits[] = {
   0x00, 0x00, 0x00, 0x00, 0xc0, 0x07, 0x80, 0x0f, 0x80,
   0x1f, 0xfc, 0x3f, 0xfc, 0x7f, 0xfc, 0xff, 0xfc, 0x7f,
   0xfc, 0x3f, 0x80, 0x1f, 0x80, 0x0f, 0xc0, 0x07, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00};
```

처음 두 행은 화소로 너비와 높이를 결정한다. 다음 두 행은 열점의 자리표를 지정한다. 열점은 비트매프가 마우스유표로 사용할 때 마우스위치로 리용되는 비트매프안의 정확한 x와 y화소위치이다. 열점의 지정은 자유이므로 두 행을 생략할수 있다. 그림 9-5는 비트매프의 모양을 보여준다. 열점은 오른쪽 화살의 끝점에 있다.

그림 9-5. 흑색과 백색의 도형을 정의하는 비트매프

비트설정은 파일에서 바이트값으로 씌여지며 매개 수는 8개 화소의 on 또는 off상태를 지정한다. 화소들은 처음에 왼쪽에서 오른쪽으로, 그다음 우에서 아래로 사영된다. 그것들은 모두 하나의 배렬에 보관되므로 그것을 사용하는 쏘프트웨어는 선분들이 끝나는곳을 알기 위하여 높이와 너비정보를 가지고 있어야 한다.

제7절. 비프매프편의프로그람

비트매프파일들을 만들고 그것들을 수정하는데 사용할수 있는 편의프로그람이 있다. 16×16의 기정크기를 가지는 새 비트매프를 만들 때에는 인수없이 지령이름을 입력한다. 24×32화소크기의 새 비트매프를 만들려고 한다면 다음과 같이 지령을 입력한다.

bitmap -size 24x32

일단 비트매프가 디스크에 만들어지고 써넣은 다음 편집하려면 지령행에서 다음과 같이 입력하여 적재하고 편집할수 있다.

bitmap arrow.xbm

화살편집에 사용된 창문을 그림 9-6에 보여준다.

조종단추들의 배렬로부터 알수 있는것처럼 그림을 편집하는 방법은 많다. 그림배치는 오른쪽 살창에 현시되고 왼쪽마우스단추를 사용하여 화소값을 0으로 설정할수 있고 오른쪽마우스단추로 그것들을 1로 설정할수 있다. 오른쪽의 다이야몬드모양화소는 열점를 가리키며 1개 열점만 있을수 있다. 열점을 설정하기 위하여 Set Hot Spot단추를 누른 다음 화소를 선택한다.
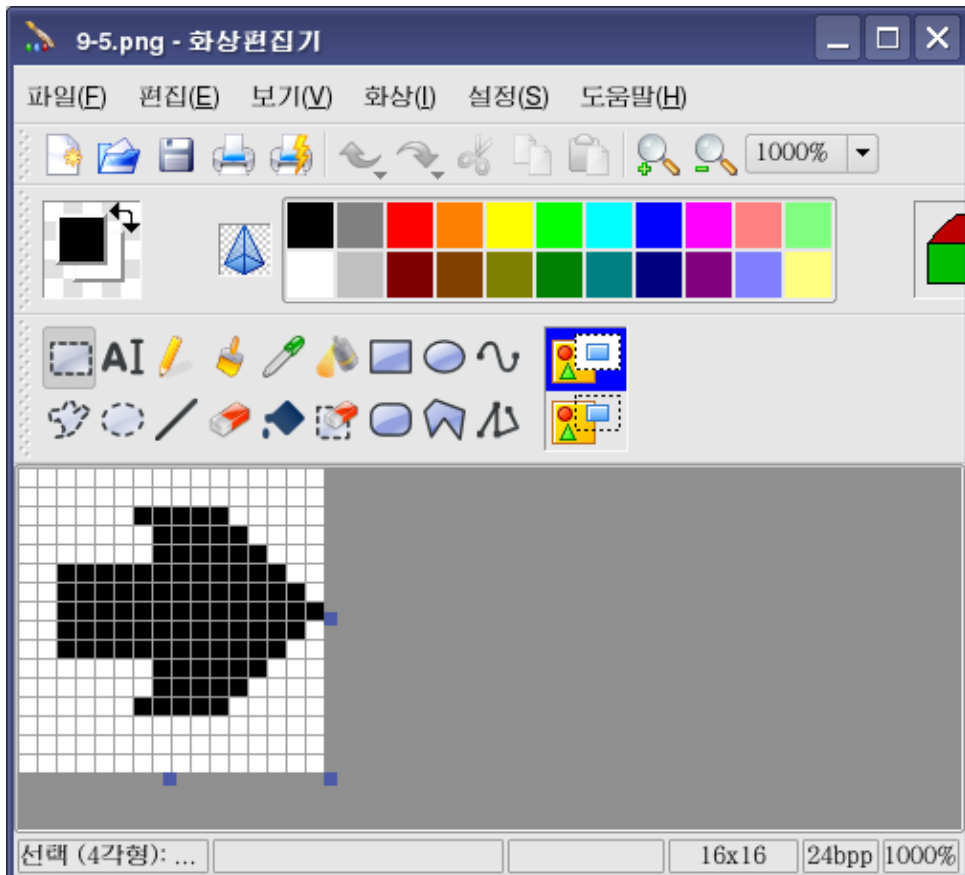
그림 9-6. arrow.xbm을 적재한 비트매프편집기

비트매프편의프로그람은 유표를 만드는데 쓰인다. 유표는 2개 비트매프를 요구하는데 하나는 유표용, 다른 하나는 마스크용이다. 유표를 만드는 과정은 8장에서 설명하였다.

알아두기: 이 프로그람은 표준X11배포물의 한 부분이다. 단추들과 차림표표식자들은 이 프로그람이 전혀 다른 창문부품과 편의프로그람들을 리용하여 개발된것이므로 KDE의것들과 완전히 다르다. 기초로 하고있는 X11표준과 규약원안은 전혀 다른 쏘프트웨어에 기초하는 모든 프로그람들을 동시에 같은 현시기에서 실행할수 있게 한다.

제8절. 차림표와 도구띠용으로 도형을 사용자정의하기

도구띠의 그림기호는 항목을 사용할수 없거나 또는 도구띠단추 창문부품이 현재 사용자에 의해 마우스단추로 선택되였다는것을 보여주기 위하여 변경할수 있다. 단추의 상태를 보여주기 위하여 픽스매프의 형태를 변경할 필요가 있다. 그러기 위하여 QIconSet클라스는 하나의 QPixmap를 입력으로 받아들이고 2개의 다른 크기로 3개 픽스매프를 생성한다. 6개의 픽스매프는 6장에서 설명한것처럼 도구띠와 차림표에서 사용될수 있다.

다음 실례프로그람은 그림 9-7과 같이 도형파일의 선택과 그 도형의 6가지 각이한 형식을 현시한다. 5장에서 설명한 QFileDialog는 도형파일로부터 QPixmap를 선택하고 적재하는데 사용된다. 그때 QIconSet객체는 그림에 보여준 픽스매프의 6개 판을 만드는데 사용된다.

그림 9-7. 6개의 각이한 형식의 그림기호

SetIcon머리부파일

```
1 /* seticon.h */
2 #ifndef SETICON_H
3 #define SETICON_H
4
5 #include <qwidget.h>
6 #include <qlayout.h>
```

```
7 #include <qlabel.h>
8 #include <qpixmap.h>
9 #include <qpushbutton.h>
10
11 class SetIcon: public QWidget
12 {
13     Q_OBJECT
14 public:
15     SetIcon(QWidget *parent=0,const char *name=0);
16 private:
17     QVBoxLayout *makeVerticalBox();
18     QGridLayout *makeGrid();
19     void insertNewPixmap();
20 private:
21     QPixmap pixmap;
22     QString pixmapName;
23     QPushButton *button;
24     QLabel *picLabel;
25     QLabel *nameLabel;
26     QLabel *normal;
27     QLabel *disabled;
28     QLabel *active;
29     QLabel *small;
30     QLabel *large;
31     QLabel *normalSmall;
32     QLabel *normalLarge;
33     QLabel *disabledSmall;
34     QLabel *disabledLarge;
35     QLabel *activeSmall;
36     QLabel *activeLarge;
37 public slots:
38     void newPixmap();
39 };
40
41 #endif
```

17～19행에서 3개의 내부메쏘드들이 선언된다. 메쏘드 makeVerticalBox()와 makeGrid()는 최상위창문을 배치하기 위하여 구성자에서 사용한다. 메쏘드 insertNewPixmap()는 새로운 QPixmap를 만들어 현시할 필요가 있을 때마다 호출된다.

21행과 22행의 QPixmap와 QString은 현재 픽스매프와 그 이름을 보관한다.

23～36행에서 선언된 누름단추와 표식자들은 현시기에 나타나는것들이다. picLabel과 nameLabel이라는 표식자들은 적재된후 수정안된 픽스매프와 파일이름을 현시한다. 26～30행의 표식자들은 그림 9-7에 보여준 표를 설명하는데 사용되며 31～36행의 표식자들은 픽스매프의 6개 판의 매개를 현시하는데 사용된다.

SetIcon

```
1 /* seticon.cpp */
2 #include <kapplication.h>
3 #include <qfiledialog.h>
4 #include "seticon.h"
5
6 int main(int argc, char **argv)
7 {
8     KApplication app(argc, argv,  "seticon");
9     SetIcon seticon;
10    seticon.show();
11    app.setMainWidget(&seticon);
12    return  (app.exec());
13 }
14
15 SetIcon::SetIcon(QWidget *parent,const char *name)
16    : QWidget(parent,name)
17 {
18    pixmapName = "hil-app-go.png";
19    pixmap = QPixmap(pixmapName);
20
21    QHBoxLayout *hbox = new QHBoxLayout(this,5);
22    QVBoxLayout *vbox = makeVerticalBox();
23    hbox->addLayout(vbox);
24    hbox->addSpacing(50);
25    QGridLayout *grid = makeGrid();
26    hbox->addLayout(grid);
```

```
27    hbox->activate();
28
29    insertNewPixmap();
30
31    connect(button,SIGNAL(clicked()),
32            this,SLOT(newPixmap()));
33 }
34
35 QVBoxLayout *SetIcon::makeVerticalBox()
36 {
37    QVBoxLayout *vbox = new QVBoxLayout(5);
38
39    vbox->addStretch(1);
40
41    button = new QPushButton("Select",this);
42    button->setFixedSize(button->sizeHint());
43    vbox->addWidget(button);
44
45    vbox->addStretch(1);
46
47    picLabel = new QLabel("",this);
48    picLabel->setAutoResize(TRUE);
49    picLabel->setAlignment(AlignHCenter | AlignVCenter);
50    vbox->addWidget(picLabel);
51
52    nameLabel = new QLabel("",this);
53    nameLabel->setAutoResize(TRUE);
54    nameLabel->setAlignment(AlignHCenter | AlignVCenter);
55    vbox->addWidget(nameLabel);
56
57    vbox->addStretch(1);
58
59    return  (vbox);
60 }
61 QGridLayout *SetIcon::makeGrid()
```

```
62 {
63     QGridLayout *grid = new QGridLayout(4,3);
64
65     normal = new QLabel("Normal",this);
66     grid->addWidget(normal,1,0);
67     disabled = new QLabel("Disabled",this);
68     grid->addWidget(disabled,2,0);
69     active = new QLabel("Active",this);
70     grid->addWidget(active,3,0);
71     small = new QLabel("Small",this);
72     grid->addWidget(small,0,1);
73     large = new QLabel("Large",this);
74     grid->addWidget(large,0,2);
75
76     normalSmall = new QLabel("",this);
77     grid->addWidget(normalSmall,1,1);
78     normalLarge = new QLabel("",this);
79     grid->addWidget(normalLarge,1,2);
80     disabledSmall = new QLabel("",this);
81     grid->addWidget(disabledSmall,2,1);
82     disabledLarge = new QLabel("",this);
83     grid->addWidget(disabledLarge,2,2);
84     activeSmall = new QLabel("",this);
85     grid->addWidget(activeSmall,3,1);
86     activeLarge = new QLabel("",this);
87     grid->addWidget(activeLarge,3,2);
88
89     return  (grid);
90 }
91 void SetIcon::insertNewPixmap()
92 {
93     picLabel->setPixmap(pixmap);
94     nameLabel->setText(pixmapName);
95
96     QIconSet iconset(pixmap);
```

```
97
98   QPixmap p;
99   p = iconset.pixmap(QIconSet::Small,QIconSet::Normal);
100  normalSmall->setPixmap(p);
101  p = iconset.pixmap(QIconSet::Large,QIconSet::Normal);
102  normalLarge->setPixmap(p);
103
104  p = iconset.pixmap(QIconSet::Small,QIconSet::Disabled);
105  disabledSmall->setPixmap(p);
106  p = iconset.pixmap(QIconSet::Large,QIconSet::Disabled);
107  disabledLarge->setPixmap(p);
108
109  p = iconset.pixmap(QIconSet::Small,QIconSet::Active);
110  activeSmall->setPixmap(p);
111  p = iconset.pixmap(QIconSet::Large,QIconSet::Active);
112  activeLarge->setPixmap(p);
113 }
114 void SetIcon::newPixmap()
115 {
116  QString filter = "Icon (*.png *.xpm *.xbm)";
117  QString name = QFileDialog::getOpenFileName("",
118          filter,this);
119  if(!name.isEmpty()) {
120          int length = name.length() - name.findRev('/');
121          pixmapName = name.right(length - 1);
122          pixmap = QPixmap(name);
123          insertNewPixmap();
124  }
125 }
```

15행의 선두에 SetIcon창문부품의 구성자가 있고 11행에서 응용프로그람의 최상위창문으로 사용된다. 18행과 19행은 초기픽스매프의 이름과 값을 지정한다. 창문은 왼쪽에 vbox라는 QVBoxLayout, 오른쪽에 grid라는 QGridLayout를 포함하는 수평칸 hbox로서 배치된다. 22행과 25행의 makeVerticalBox()와 makeGrid()호출은 수평칸에 포함된 2개 보조배치를 만들어낸다.

29행의 insertNewPixmap()호출은 초기픽스매프를 현재 현시하려는것으로서 설치한다. 31

행의 connect()호출은 단추를 누를 때마다 처리부메쏘드 newPixmap()를 실행하도록 설정한다.

35행의 메쏘드 makeVerticalBox()는 Select단추, 변경되지 않은 도형을 현시하기 위한 현시표식자 그리고 도형파일의 이름을 보관하는 표식자를 만든다. 이것들은 모두 수직칸에 삽입된다. 41～43행에서 단추를 만들어서 삽입한다. 47～55행에서 2개 표식자를 만들어 수직칸에 추가한다. 표식자들은 비여놓는데 픽스매프와 그 파일이름은 후에 설치된다. 61행에서 시작하는 메쏘드 makeGrid()는 현재 픽스매프의 여러가지 모양을 현시하는데 쓰이는 QLabel객체들의 표를 만들기 위하여 QGridLayout를 사용한다. 살창은 3×4세포 크기이다. 첫행과 첫렬은 그림 9-7에서 창문의 오른쪽에서 볼수 있는것처럼 주석표식자들에 사용된다. 65～74행에서는 본문을 리용하여 주석으로서 표식자들을 만든다. 76～87행에서 만든 표식자들은 픽스매프도형들을 현시하기 위한것이므로 본문없이 만들어진다.

91행에서 메쏘드 insertNewPixmap()는 현시기에 현재 픽스매프정보를 채운다. 이 메쏘드는 프로그람이 처음 기동할 때 기정픽스매프를 설치하기 위해서만 호출되며 새 픽스매프가 선택될 때마다 다시 한번 호출된다.

객체의 pixmap마당에 보관된 픽스매프를 사용하여 96행에서 QIconSet객체 iconset을만든다. 지금 요구되는것은 6개의 수정된 픽스매프를 현시할 표식자창문부품들을 얻어서 삽입하는것이다. 메쏘드 pixmap()는 도형의 매개 판을 얻는데 사용된다. 메쏘드에 넘어오는 인수는 6개중 어느판을 돌려주는가를 결정한다. 첫 인수는 돌려주는 픽스매프가 Small이나 Large라는것을 지정한다. 둘째 인수는 Normal, Disabled이나 Active이라는것을 지정한다. 인수값들은 QIconSet클라스에서 렬거형으로 정의된다.

선택한 픽스매프가 작은 그림기호와 꼭같은 크기이면 Small판은 원래것과 다름없고 확장판은 Large용으로 만들어진다. 다른 한편 픽스매프가 Large그림기호와 같은 크기이면 Small그림기호가 생성된다. 크기들은 절대크기로 조절되지 않고 도형의 원래크기에 관계된다. 실례로 매우 큰 도형을 선택한다면 그 크기는 Large그림기호용으로 변경되지 않고 오직 Small그림기호에 대해서만 축소된다.

114행의 처리부메쏘드 newPixmap()는 Select단추를 찰칵할 때마다 호출된다. 그것은 117행에서 도형파일을 선택하기 위하여 getOpenFileName()호출에 의해 QfileDialog를 펼친다. 116행에서 정의한 려과기는 .png, .xpm 그리고 .xbm확장자를 가진 파일들로 제한하지만 필요하다면 다른 도형파일들을 포함할수 있다. 파일이름이 선택된다면 그 완전경로이름이 되돌아온다. 120행의 QString의 메쏘드 findRev()와 length()은 경로없이 파일이름의 길이를 결정하는데 사용되며 121행의 right()호출은 현시하려는 파일의 이름을 꺼낸다. 122행에서 새로운 픽스매프를 만들고 123행에서 insertNewPixmap()호출에 의해 새로운 현시가 구성된다.

요 약

도형처리에는 많은것이 있으나 이 장에서는 차림표와 도구띠용 도형단추들을 만들수 있는 충분한 정보를 설명하였다. 이 장에서는 다음과 같은 내용을 설명하였다.

• 도형의 2가지 기본형식은 비트매프와 픽스매프이다. 비트매프는 색정보를 담고있지 않으며 매개 화소에 1 또는 0을 지정한다. 픽스매프는 임의의 색을 가질수 있다. 그리고 픽스매프(XPM형식)와 비트매프(XBM형식) 모두는 프로그람으로 직접 콤파일하거나 실행시에 동적으로 적재할수 있다.

• QWidget를 계승하는 객체는 그 배경에 고정색이 아니라 픽스매프를 현시할 능력을 가지고있다.

• KDE는 많은 도형파일형식을 인식하며 쏘프트웨어는 다른것들을 후에 추가할수 있도록 설계된다. 사실상 그것들을 공유서고에 추가하여 다시 콤파일하지 않고 프로그람에서 사용할수 있다.

이 장에서 소개된 픽스매프들은 보통 사용자에게 어느 단추가 무엇을 하는가를 알리는 식별자로 리용된다. 그러나 대체로 본문과만 작업한다. 다음 장은 응용프로그람에 사용할수 있는 여러가지 서체와 서체묘사수법을 탐구한다.

제10장. 서체

학습내용

서체의 크기를 설정하고 변경하는 방법

X11서체이름달기관례의 리해

서체선택창문부품에 의한 서체선택

서체측도정보의 읽기

본문위치지정

X11, Qt 그리고 KDE에서 사용하는 서체들은 처음에 삭갈리기 쉽다. 그러나 일단 무엇을 하는가 알면 매우 간단하다. 매우 유연하고 직접적인 서체취급방법이 제안되였다. 응용프로그람에서는 자기가 특별히 좋아하는 서체를 정확히 사용할수 있다. 또한 서체선택에서 시간손실을 없앨수 있고 체계가 선택기준에 맞는 서체를 선택하도록 할수 있다. 또한 사용자가 서체를 선택할수 있는 2개의 창문부품이 있다.

서체는 크기와 형태에 따라서 각이하다. 서체에 적용되는 특수한 측도들이 있다. 그 표준값들을 사용하여 현시기에 문자위치를 지정함으로써 모든 서체들을 같은 방법으로 취급할수 있다. 이 장은 각이한 형과 크기의 서체의 얻기, 위치지정 그리고 묘사방법을 론의한다.

제1절. 서체의 해부

서체에서 한 문자 또는 한개 문자렬에 대하여 수많은 측도를 설정할수 있다. 문제를 복잡하게 하는것은 일부 문자들이 다른 문자들보다 크고 일부는 다른 문자들보다 아래로 내려가며 일부 문자들은 다른 문자들보다 넓다는 사실이다. 또한 한 문자렬안에서 서로 린접문자들에서 한 문자가 다른 문자와 겹칠수 있으며 이것은 큰 문자의 웃끝이 오른쪽으로 늘어나는 경사체에서 일반적으로 볼수 있다.

그림 10-1에서는 문자를 구성하는 측도들을 보여준다. 원점은 문자를 그리는데 사용되는 x와 y자리표점이다. 다시 말하면 문자를 특정한 자리표점에 그릴 때 실제로 그것은 그 점의 오른쪽우에 나타난다. 다른 한편 문자 p는 점의 오른쪽에 나타나지만 그 우와 아래에 모두 나타난다. 문자의 화소묘사 또는 도형설계를 도형문자설계(글리프)라고 한다. 모든 글리프는 원점에 준하여 설계된다. 원점은 문자자체를 그리는데 필요하다. 이 원점들의 렬을 기준선이라고 부른다. 상승값과 하강값들은 기준선으로부터 문자의 웃끝 그리고 아래끝까지의 치수이다. 상승과 하강의 총합은 문자의 높이이다. 문자의 너비는 원점에서 문자의 오른변까지로 계산된다. 왼쪽위치(lbearing)는 원점에서 문자까지 거리이고 오른쪽위치(rbearing)는 문자의 도형부분의 너비이다.
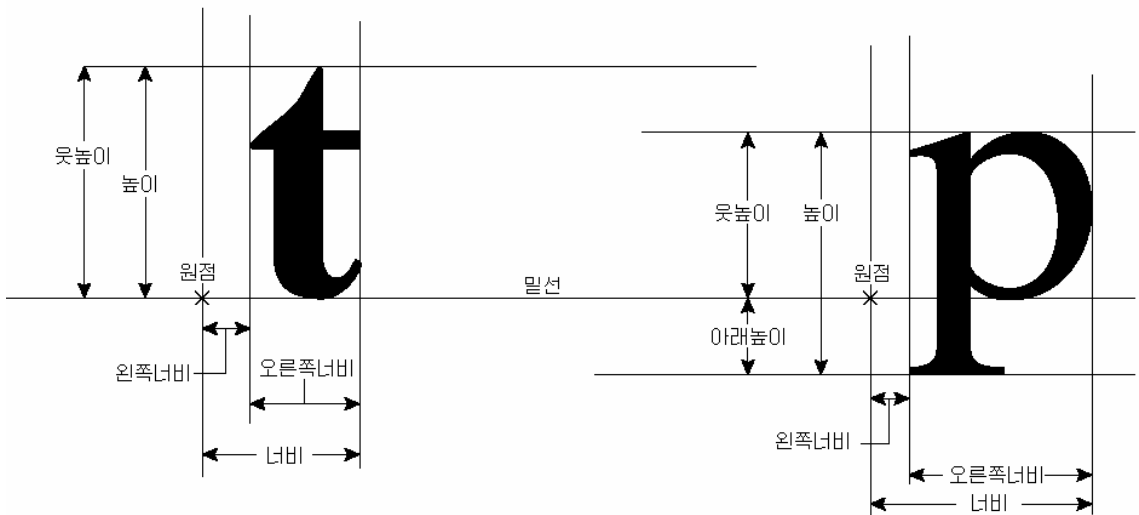
그림 10-1. 한 문자에서 서체측도

그림 10-1에서 상승, 하강 그리고 높이치수들로부터 하강값이 0이라는것을 알수 있다. 또한 왼쪽위치값이 0일수도 있다. 사실상 원점의 왼쪽에 그려지는 문자의 글리프인 경우에는 심지어 부수일수 있다. (실례로 경사서체의 밑부분)

그림 10-2는 문자렬을 구성하는 측도들을 보여준다. 상승, 하강과 높이는 최대범위밖으로 우로, 아래로, 혹은 우아래로 늘어난 유도구역을 포함할수 있다. 너비도 갈다. 문자렬둘레의 경계는 프로그람이 본문(다른 서체들의 본문에서도)부분들 쉽게 배치하고 정확한 간격을 둘수 있게 한다. 문자렬의 원점 즉 문자렬을 그리는데 리용되는 x와 y자리표는 문자렬에서 제일왼쪽문자의 원점이다. 다른 문자들의 원점은 내부적으로 문자렬을 그릴 때 매개 문자들을 배치하는데 사용된다.
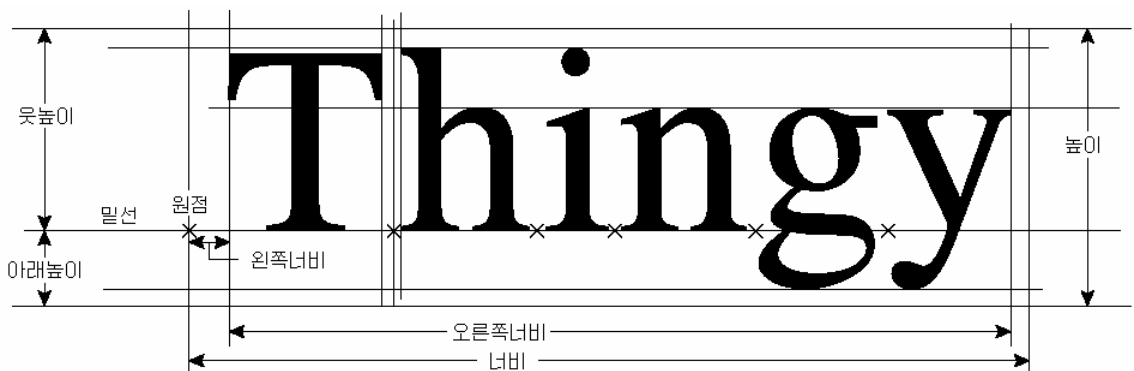
그림 10-2. 문자렬에서 서체측도

제2절. 서체의 이름

서체들은 디스크파일들에 보관된다. 서체파일들은 표준 X배포물의 한 부분이다. 응용프로그람이 서체를 요구할 때 그것은 응용프로그람이 아니라 X봉사기에서 파일형태로 적재된다. 서체들은 봉사기에 있으므로 현시를 위해 응용프로그람에서 봉사기에로 구체적인 서체정보의 넘기기로 인한 추가비용은 없다. 이 통신량감소는 다른 콤퓨터의 응용프로그람에 의해 조종되는 국부X창문(국부봉사기에서)을 여는것이 례사로운 일로 되여있으므로 많은 시간을 절약할수 있다. 그밖에 2개이상의 응용프로그람이 같은 서체를 사용하고있다면 오직 한개 사본이 적재되여야 한다.

서체파일들은 대체로 /usr/lib/X11/fonts의 보조등록부에 보관된다. 매개 보조등록부에 있는 서체파일들은 확장자 .pcf 또는 .pcf.gz를 가진다. 서체들과 함께 같은 등록부에는 별명이름들을 실제서체파일이름들로 넘기는 fonts.dir라는 파일이 있다. fonts.dir에서 정의한 서체들에 별명을 할당하는데 쓰이는 fonts.alias도 있다. 실례로 다음과 같이 그 이름을 지정함으로써 10x20.pcf.gz라는 서체파일을 사용할수 있다.

10x20

혹은 그것에 할당된 별명을 사용할수 있다. 즉

-misc-fixed-medium-r-normal--20-200-75-75-c-100-iso8859-1

간략형식은 기억하기 쉽고 긴 형식은 리해하기 쉽다. 그리고 쏘프트웨어는 이름의 매개부분에 대리기호들을 사용하여 선택할수 있게 한다. 이름의 매개 부분은 특별한 의미를 가진다. 표 10-1은 그림 10-3에 보여준 이름의 매개 요소들을 설명한다.

그림 10-3. 서체이름의 요소들

표 10-1. 서체이름의 요소

부분이름	설명
서체제작자	서체를 만든 회사 혹은 기관이름. 몇가지 일반적인 이름들로서는 adobe, b&h, bitstream, dec, schumacher, sony 그리고 sun. 만든 기관이 그 서체를 요구하지 않는다면 이름은 misc.
서체계렬	서체가 포함된 관련서체계렬의 이름. 가능한 이름들은 lucida, times , courier, helvetica 등.
무게	한획무게. 대체로 medium이나 bold가 있고 또한 black , book , demibold , light , 혹은 regular도 있다.
경사	매개 문자의 각도는 italic, oblique, 혹은 r(roman의 략자로서 오른쪽우를 의

부분이름	설명
	미한다)일수 있다.
비례너비	높이와 너비사이의 관계는 대체로 표준이지만 그것은 응축, 반응축되거나, 좁아지거나 배로 늘일수 있다.
화소	화소단위의 서체크기. 보통 서체크기는 점으로 측정된다. (점은 1in의 1/72이다.) 화소크기값을 얻기 위하여 점크기를 화소크기로 변환한다. 이것은 점크기가 화소경계를 벗어나므로 잘리울수 있다는것을 의미한다.
점	1/10점단위의 서체의 점크기. 실례에서 값 80은 이것이 8점서체라는것을 가리킨다. 점크기와 화소크기사이의 관계는 수직 및 수평dpi(in당 점수)값들에 의해 결정된다. 이 실례에서 100dpi에서 8점서체는 11의 화소크기를 가지고있다. 같은 dpi에서 12점 서체는 17의 화소크기를 가진다.
수평dpi	분해능 1in당 수평화소수. 이 값은 화소와 점크기를 계산하는데 사용된다. 또한 서체현시에서 수평수직dpi를 결정하기 위하여 수직dpi와의 비률로 사용된다.
수직dpi	분해능 1in당 수직화소수.
간격	이것은 m(등폭서체), p(비례서체) 또는 c(문자세포)일수 있다. 등폭(monospace)서체는 모든 문자들의 너비가 같은 서체이다. 비례서체는 각이한 너비의 문자를 가진다. (실례로 글자 w가 문자 i보다 폭이 넓다.) 문자세포서체는 타자기서체에서 간격유지방법에 기초한 고정너비서체이다.
화소너비	서체안의 모든 문자들의 평균너비(1/10화소).
문자모임	이것은 문자모임을 정의하는데 사용되는 표준판이다. ISO는 여러가지 언어의 자모에 포함되는 문자모임들에 대한 표준을 확립하였다.

긴서체이름들은 설명을 줄뿐아니라 서체를 탐색할 때 대리문자들을 사용할수 있는 형식으로 되여있다. 이리하여 오직 주의할 점만 지정할 필요가 있고 그 나머지를 기정으로 한다. 실례로

-*-bookman-light-r-normal--14-*-*-*-p-*-iso8859-1

이름에서 지정된 부분들은 실제 서체와 정확히 일치해야 하며 별표는 임의의 값과 대조될수 있다. 여러개의 서체들이 대조되면 처음으로 대조되는것이 돌아온다. 앞의 실례는 이 서체를 선택할수 있다.

- adobe-bookman-light-r-normal--14-135-75-75-p-82-iso8859-1

서체이름을 지정할 때 오직 필요한 부분만 지정해야 한다. 그리하여 실제 서체이름과 대조할 더 많은 기회를 준다. 서체의 지정이 서체이름과 일치하지 않으면 기정서체가 사용되는데 그것은 대체로 자기가 요구하는 서체가 아니다.

제3절. 창문부품의 서체설정

QFont객체를 만들어 창문부품이 사용하는 서체를 지정하는데 사용할수 있다. 다음 실례는 3개의 표식자를 현시하는데 그 매개는 그림 10-4와 같이 각이한 서체를 리용한다.

```
1 /* fontset.cpp */
2 #include <kapplication.h>
3 #include <qlabel.h>
4 #include <qlayout.h>
5 #include <qfont.h>
6 #include "fontset.h"
7
8 int main(int argc, char **argv)
9 {
10     KApplication app(argc, argv, "fontset");
11     FontSet fontset;
12     fontset.show();
13     app.setMainWidget(&fontset);
14     return(app.exec());
15 }
16 FontSet::FontSet(QWidget *parent,const char *name)
17     : QWidget(parent,name)
18 {
19     QVBoxLayout *vbox = new QVBoxLayout(this,10);
20
21     QLabel *label1 = new QLabel(
22             "Bold 14-point Courier",this);
23     QFont font1("Courier",14,QFont::Bold,FALSE);
24     label1->setFont(font1);
25     vbox->addWidget(label1);
26
27     QLabel *label2 = new QLabel(
28             "20-point Fixed",this);
29     QFont font2("Fixed",20,QFont::Normal,FALSE);
30     label2->setFont(font2);
31     vbox->addWidget(label2);
```

```
32
33    QLabel *label3 = new QLabel(
34            "Bold Italic 18-point Charter",this);
35    QFont font3("Charter",18,QFont::Bold,TRUE);
36    label3->setFont(font3);
37    vbox->addWidget(label3);
38 }
```

그림 10-4. 표식자들에서 서체의 설정

일단 QFont객체가 만들어지면 setFont()호출은 표식자에 그것을 설치한다. setFont()메쏘드는 QWidget클라스를 계승하는 가상메쏘드이므로 같은 방법은 본문을 현시하는 임의의 창문부품에 동작해야 한다.

QFont구성자는 서체이름을 지정하는 인수목록을 받아들인다. 인수들은 초기에 지정한 서체파일이름들과 같은 정보를 포함하지만 그것들은 형식을 리용하기 쉽게 되여있다. QFont를 만들려면 서체계렬이름, 점크기, 문자무게 그리고 서체가 경사체인가 아닌가를 지정해야 한다. QFont클라스에서 렬거형으로 정의되는 무게번호는 Light, Normal, DemiBold, Bold 그리고 Black이다.

구성자에 넘기는 인수로서 실제로 존재하지 않는 서체 실례로 12점고정경사체를 지정할수 있으나 구성자는 서체이름짓기관례에 따라 요구하는 서체에 가장 잘 일치하는것을 탐색하므로 서체탐색에서 성공한다. QFont구성자코드를 작성하기전에 이 장의 프로그람들중 하나를 사용하여 사용가능한 서체들을 열람할수 있다.

정확한 서체이름을 지정하려고 한다면 QFont객체에 대하여 메쏘드 setRawName()를 사용할수 있다. 실례로 다음 코드는 Adobe에서 만든 Utopia계렬의 경사체인 QFont객체를 만든다.

```
QFont  font;
font.setRawName("-adobe-utopia-regular-i-normal--15-140-75-75-p-79-iso8859-1");
```

제4절. QFontDialog에 의한 서체의 선택

다음 프로그람은 QFontDialog를 펼치는데 사용할수 있는 단추를 기본창문에 표시한다. 대화칸에서 서체선택은 본문과 단추의 서체를 모두 바꾸게 하며 매개 서체설명은 그 서체자체에서 제시된다. 그림 10-5에서 기본창문과 단추의 3가지의 다른 형태를 보여준다.

fontprompt <2>
-schumacher-clean-bold-r-normal--14-140-75-75-c-80-iso8859-1

fontprompt <3>
-dec-terminal-medium-r-normal--*-160-75-75-c-*-iso8859-1
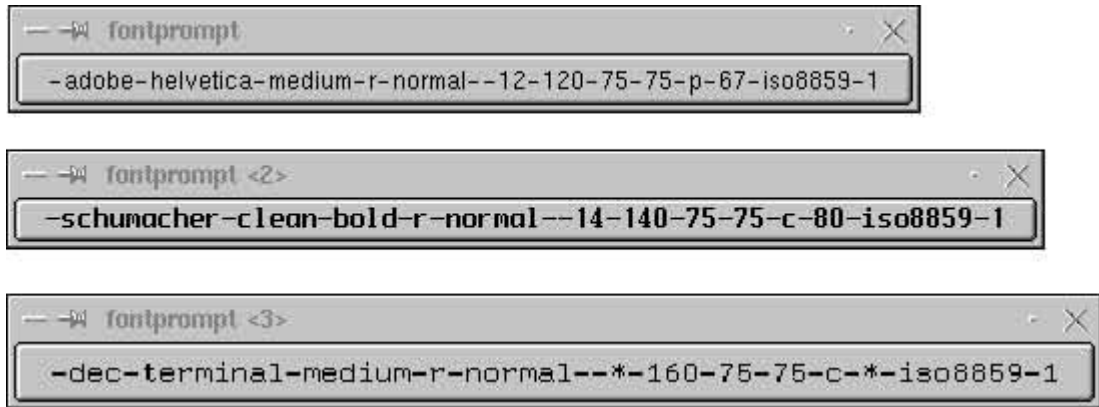
그림 10-5. 단추에 현시된 3가지 서체

FontPrompt머리부파일

```
1 /* fontprompt.h */
2 #ifndef FONTPROMPT_H
3 #define FONTPROMPT_H
4
5 #include <qwidget.h>
6 #include <qpushbutton.h>
7
8 class FontPrompt: public QWidget
9 {
10     Q_OBJECT
11 public:
12     FontPrompt(QWidget *parent=0,const char *name=0);
13 private:
14     QPushButton *button;
15 private slots:
16     void popupDialog();
17 };
18
19 #endif
```

FontPrompt클라스는 매우 간단하다. 그것은 찰칵해야 할 단추와 대화칸을 펼치기 위해 실행해야 할 처리부를 포함한다.

FontPrompt

```
1 /* fontprompt.cpp */
2 #include <kapplication.h>
3 #include <qfontdialog.h>
4 #include "fontprompt.h"
5
6 int main(int argc, char **argv)
7 {
8     KApplication app(argc, argv, "fontprompt");
9     FontPrompt fontprompt;
10    fontprompt.show();
11    app.setMainWidget(&fontprompt);
12    return(app.exec());
13 }
14 FontPrompt::FontPrompt(QWidget *parent,const char *name)
15    : QWidget(parent,name)
16 {
17    button = new QPushButton("",this);
18    QFont font = button->font();
19    button->setText(font.rawName());
20    button->setFixedSize(button->sizeHint());
21    setFixedSize(button->sizeHint());
22
23    connect(button,SIGNAL(clicked()),
24            this,SLOT(popupDialog()));
25 }
26 void FontPrompt::popupDialog()
27 {
28    bool okay;
29
30    QFont oldFont = button->font();
31    QFont newFont =
32            QFontDialog::getFont(&okay,oldFont,this);
```

```
33    if(okay) {
34            button->setFont(newFont);
35            button->setText(newFont.rawName());
36            button->setFixedSize(button->sizeHint());
37            setFixedSize(button->sizeHint());
38    }
39 }
```

14행에서 시작하는 FontPrompt구성자는 기본창문에 단추를 만들어 설치한다. 18행의 font()호출은 단추의 현재(기정)서체를 얻는다. 19행의 rawName()호출은 서체의 완전이름을 얻고 setText()를 호출하여 서체이름이 단추본문으로 사용되도록 설정한다. 20행은 setFixedSize()를 호출하여 단추크기가 본문크기와 정확히 같아지게 한다. 21행의 setFixedSize()호출은 단추에 적합하게 기본창문의 크기를 설정한다.

단추는 26행에서 처리부메쏘드에 련결된다. 기정본문을 보여주는 결과창문은 초기에 그림 10-5의 꼭대기에 보여준것과 같다.

26행의 popupDialog()처리부메쏘드는 새로운 서체를 얻고 설치하는 모든 작업을 한다. 30행의 font()호출은 단추로부터 현존서체를 얻는다. 32행의 getFont()호출은 기정으로 리용하는 현존서체를 대화칸에 넘긴다. 기정서체를 현시하는 초기창문은 그림 10-6에 보여준것과 같다.
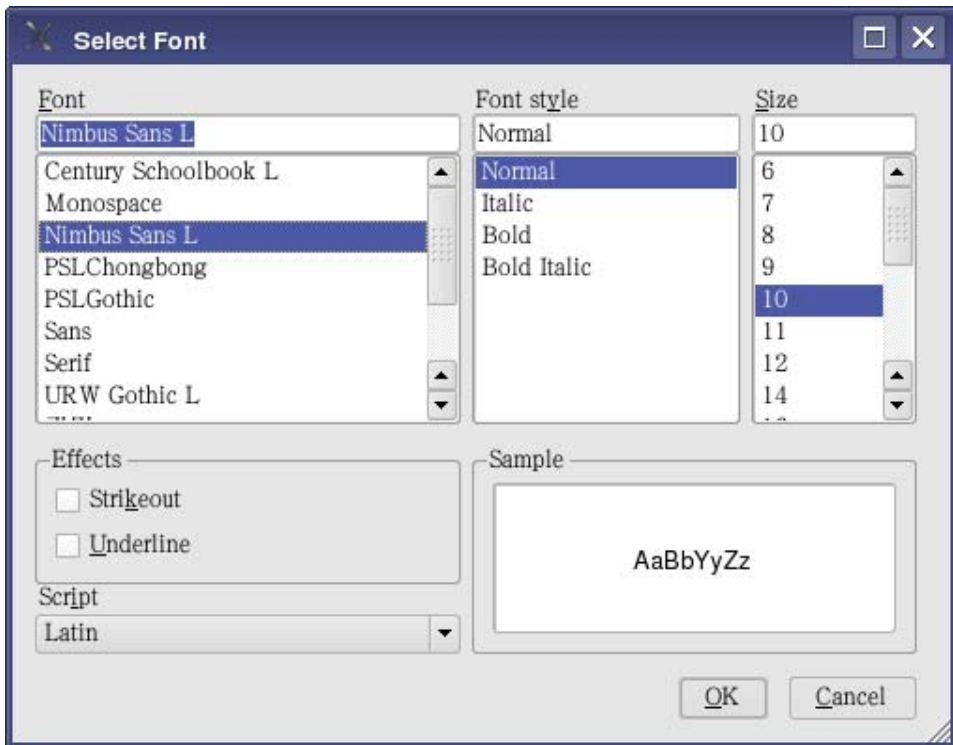
그림 10-6. 기정서체를 보여주는 서체선택

알아두기: 대화칸에서 여러개의 선택항목이 표준서체정의의 요소로 포함되지 않는다. 취소선과 밑줄용 절환단추들은 QFont클라스에 의해 추가된 2개 서체선택들이므로 대화칸의 부분이다.

OK단추를 찰칵하면 31행에서 새로운 서체가 newFont로서 등록된다. 또한 Boolean변수 okay는 TRUE로 설정된다. 34행에 현시를 갱신하기 위한 setFont()호출이 있다. setFont()메쏘드는 QWidget클라스의 성원함수로서 선언되며 이것은 본문을 현시하는 창문부품의 본문을 설정하는데 사용된다는것을 의미한다. 35행은 rawName()과 setText()를 호출하여 단추에 새로운 서체의 완전서술을 삽입한다. 36행과 37행은 서체변경이 늘 창문크기를 변경하므로 단추와 창문의 크기를 조절할 필요가 있다.

제5절. KFontDialog에 의한 서체의 선택

다음의 서체선택프로그람은 KFontDialog를 서체선택에 사용하는것을 제외하면 앞의 실례와 거의 같다. 대화칸은 그림 10-7에 보여준다.
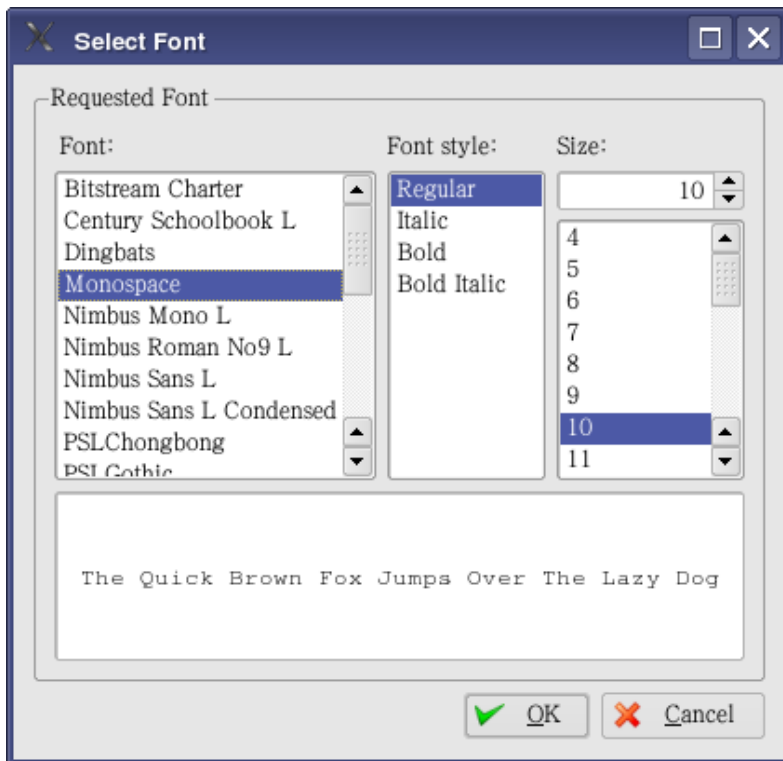
그림 10-7. 서체선택대화칸

FontPrompt2머리부파일

```
1 /* fontprompt2.h */
2 #ifndef FONTPROMPT2_H
3 #define FONTPROMPT2_H
4
```

```
5 #include <qwidget.h>
6 #include <qpushbutton.h>
7
8 class FontPrompt2: public QWidget
9 {
10     Q_OBJECT
11 public:
12     FontPrompt2(QWidget *parent=0,const char *name=0);
13 private:
14     QPushButton *button;
15 private slots:
16     void popupDialog();
17 };
18
19 #endif
```

FontPrompt2

```
1 /* fontprompt2.cpp */
2 #include <kapplication.h>
3 #include <kfontdialog.h>
4 #include "fontprompt2.h"
5
6 int main(int argc, char **argv)
7 {
8      KApplication app(argc, argv, "fontprompt2");
9      FontPrompt2 fontprompt2;
10     fontprompt2.show();
11     app.setMainWidget(&fontprompt2);
12     return(app.exec());
13 }
14 FontPrompt2::FontPrompt2(QWidget *parent,const char *name)
15     : QWidget(parent,name)
16 {
17     button = new QPushButton("",this);
18     QFont font = button->font();
```

```
19    button->setText(font.rawName());
20    button->setFixedSize(button->sizeHint());
21    setFixedSize(button->sizeHint());
22
23    connect(button,SIGNAL(clicked()),
24            this,SLOT(popupDialog()));
25 }
26 void FontPrompt2::popupDialog()
27 {
28    QFont font = button->font();
29    int result = KFontDialog::getFont(font);
30    if(result == QDialog::Accepted) {
31            button->setFont(font);
32            button->setText(font.rawName());
33            button->setFixedSize(button->sizeHint());
34            setFixedSize(button->sizeHint());
35    }
36 }
```

14행에서 시작하는 FontPrompt2구성자는 한개 단추를 가진 창문을 만든다. 단추의 모양은 18행과 19행에서 자기의 서체이름을 보여주기 위해 초기화된다. 20행과 21행은 제일웃준위창문과 단추를 본문크기로 조절한다.

26행의 처리부메쏘드 popupDialog()는 단추를 찰칵할 때마다 호출된다. 29행의 정적메쏘드 getFont()는 대화칸을 펼치고 선택하였으면 값 QDialog::Accepted를 돌려주며 혹은 선택하지 않았으면 QDialog::Rejected를 돌려준다. 31행과 32행은 선택된 서체를 사용하도록 단추를 갱신하고 단추의 본문을 서체의 설명이름으로 설정한다. 32행과 33행은 단추와 창문을 본문에 알맞게 조절한다.

대화칸 KFontDialog와 QFontDialog는 아주 류사하다. 차이점은 QFontDialog는 2개의 검사칸(밑선과 취소선)을 추가적으로 더 가지고있으며 KFontDialog는 창문의 아래에 실례본문을 현시하는것이다. 이것들은 모두 문자렬을 편집하여 사용하려는 본문에 어떤 서체가 좋은가를 알수 있게 하지만 KFontDialog는 다만 본문을 초기화하여 프로그람에 돌려주게 한다. 그러기 위하여 다음과 같이 대화칸을 만든다.

```
int result = KfontDialog::getFontAndText(font, btext);
KfontDialog::getFontAndText(font, btext);.
```

btext인수는 OK단추를 찰칵할 때마다 대화칸창문의 아래에 현시된 본문을 받아들이는 QString객체이다.

제6절. 측도에 의한 서체배치

측도값들은 창문에서 서체들을 배치하는데 사용할수 있다. 다음 응용프로그람은 서체를 선택하고 서체의 측도들로부터 계산된 위치에 현시되도록 한다.

그림 10-8에서는 서체를 선택하고 본문위치를 지정하는데 쓰이는 단추들을 가진 프로그람의 제일웃준위창문을 보여준다. 제일 우의 3개의 단추는 검은 직4각형안에 본문을 수직방향으로 배치하며 아래에 있는 3개 단추는 본문을 수평방향으로 배치한다. 아래에 있는 큰 단추는 그림 10-6에서 보여준것처럼 대화칸을 펼친다. 이 대화칸은 서체를 바꾸는데 사용할수 있다.
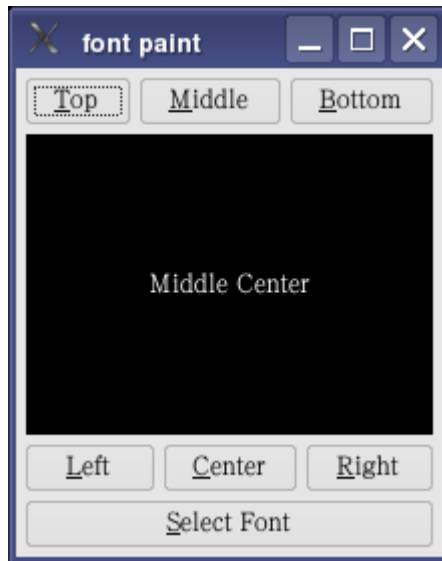
그림 10-8. 창문안에서 본문의 배치

FontPaint머리부파일

```
1 /* fontpaint.h */
2 #ifndef FONTPAINT_H
3 #define FONTPAINT_H
4
5 #include <qwidget.h>
6 #include <qframe.h>
7 #include <qpushbutton.h>
8
9 class FontPaint: public QWidget
10 {
11     Q_OBJECT
12 public:
```

```
13     FontPaint(QWidget *parent=0,const char *name=0);
14 private:
15     void updateDisplay();
16 protected:
17     void paintEvent(QPaintEvent *);
18 private:
19     enum { Hleft, Hcenter, Hright };
20     enum { Vtop, Vmiddle, Vbottom };
21     int Hposition;
22     int Vposition;
23     QPushButton *topButton;
24     QPushButton *middleButton;
25     QPushButton *bottomButton;
26     QPushButton *leftButton;
27     QPushButton *centerButton;
28     QPushButton *rightButton;
29     QPushButton *selectFontButton;
30     QWidget *frame;
31     QFont font;
32 private slots:
33     void popupDialog();
34     void setTop() { Vposition = Vtop;
35             updateDisplay(); }
36     void setMiddle() { Vposition = Vmiddle;
37             updateDisplay(); }
38     void setBottom() { Vposition = Vbottom;
39             updateDisplay(); }
40     void setLeft() { Hposition = Hleft;
41             updateDisplay(); }
42     void setCenter() { Hposition = Hcenter;
43             updateDisplay(); }
44     void setRight() { Hposition = Hright;
45             updateDisplay(); }
46 };
47
```

```
48 #endif
```

FontPaint클라스는 최상위창문으로 사용되는 창문부품이다. 19～20행의 렬거형들은 21～22행에서 Hposition과 Vposition에 기억된 현재 상태를 가지고 본문의 수직 및 수평위치를 지정하는데 사용된다. 23～28행에서 선언된 누름단추들은 각각 Vposition과 Hposition에 값을 보관한다. 34～45행의 매개 처리부메쏘드들은 하나의 단추에 련결되며 메쏘드가 호출될 때 그것은 본문의 위치를 갱신하고 updateDisplay()를 호출하여 창문을 그린다.

FontPaint

```
1 /* fontpaint.cpp */
2 #include <kapplication.h>
3 #include <qfontdialog.h>
4 #include <qpainter.h>
5 #include <qlayout.h>
6 #include "fontpaint.h"
7
8 int main(int argc, char **argv)
9 {
10    KApplication app(argc, argv, "fontpaint");
11    FontPaint fontpaint;
12    fontpaint.show();
13    app.setMainWidget(&fontpaint);
14    return  (app.exec());
15 }
16 FontPaint::FontPaint(QWidget *parent,const char *name)
17    : QWidget(parent,name)
18 {
19    QHBoxLayout *hbox;
20    QVBoxLayout *vbox = new QVBoxLayout(this,5);
21
22    hbox = new QHBoxLayout(5);
23    topButton = new QPushButton("Top",this);
24    hbox->addWidget(topButton);
25    connect(topButton,SIGNAL(clicked()),
26            this,SLOT(setTop()));
27    middleButton = new QPushButton("Middle",this);
28    hbox->addWidget(middleButton);
```

```
29    connect(middleButton,SIGNAL(clicked()),
30            this,SLOT(setMiddle()));
31    bottomButton = new QPushButton("Bottom",this);
32    hbox->addWidget(bottomButton);
33    connect(bottomButton,SIGNAL(clicked()),
34            this,SLOT(setBottom()));
35    vbox->addLayout(hbox);
36
37    frame = new QWidget(this);
38    frame->setMinimumSize(150,150);
39    vbox->addWidget(frame);
40
41    hbox = new QHBoxLayout(5);
42    leftButton = new QPushButton("Left",this);
43    hbox->addWidget(leftButton);
44    connect(leftButton,SIGNAL(clicked()),
45            this,SLOT(setLeft()));
46    centerButton = new QPushButton("Center",this);
47    hbox->addWidget(centerButton);
48    connect(centerButton,SIGNAL(clicked()),
49            this,SLOT(setCenter()));
50    rightButton = new QPushButton("Right",this);
51    hbox->addWidget(rightButton);
52    connect(rightButton,SIGNAL(clicked()),
53            this,SLOT(setRight()));
54    vbox->addLayout(hbox);
55
56    selectFontButton = new QPushButton("Select Font",this);
57    vbox->addWidget(selectFontButton);
58    connect(selectFontButton,SIGNAL(clicked()),
59            this,SLOT(popupDialog()));
60
61    Hposition = Hcenter;
62    Vposition = Vmiddle;
63    font = frame->font();
```

```
64     updateDisplay();
65 }
66 void FontPaint::popupDialog()
67 {
68     bool okay;
69
70     QFont newFont = QFontDialog::getFont(&okay,font,this);
71     if(okay) {
72             font = newFont;
73             updateDisplay();
74     }
75 }
76 void FontPaint::updateDisplay()
77 {
78     int x;
79     int y;
80     QString text;
81     QPainter painter(frame);
82     painter.setFont(font);
83     QFontMetrics fm = painter.fontMetrics();
84
85     painter.setBackgroundColor(QColor("black"));
86     painter.setPen(QColor("white"));
87
88     QRect rect = painter.window();
89     painter.eraseRect(rect);
90
91     switch(Vposition) {
92             case Vtop:
93                     y = fm.ascent();
94                     text = "Top ";
95                     break;
96             case Vmiddle:
97                     y = rect.height() / 2;
98                     y += (fm.ascent() - fm.descent()) / 2;
```

```
99                  text = "Middle ";
100                 break;
101           case Vbottom:
102                 y = rect.height() - fm.descent();
103                 text = "Bottom ";
104                 break;
105   }
106   switch(Hposition) {
107           case Hleft:
108                 x = 0;
109                 text += "Left";
110                 break;
111           case Hcenter:
112                 text += "Center";
113                 x = (rect.width() - fm.width(text)) / 2;
114                 break;
115           case Hright:
116                 text += "Right";
117                 x = rect.width() - fm.width(text);
118                 break;
119   }
120   painter.drawText(x,y,text);
121 }
122 void FontPaint::paintEvent(QPaintEvent *)
123 {
124   updateDisplay();
125 }
```

16행에서 시작하는 FontPaint구성자는 수직칸을 초기배치로 사용하며 19～59행에서 거기에 창문부품들을 채운다. 3개 단추를 지정하는 두 행은 각각 수평칸에 놓인다. 6개의 위치단추는 fontpaint.h에서 선언된 처리부메쏘드중 하나와 각각 련결된다. 37행에서 만든 frame창문부품은 그림 10-8에 보여준것처럼 본문을 현시하는데 쓰이는 검은 직4각형이다. 56～59행에서 만들어진 제일 아래 단추는 popupDialog()라는 처리부메쏘드에 련결된다.

66행에서 정의된 popupDialog()처리부는 QFontDialog클라스에서 정적메쏘드 getFont()를 사용하여 새로운 서체를 얻는다. 새로운 서체가 선택되면 72행에서 그것이 현재서체로 되며 updateDisplay()호출이 이루어져 새로운 틀창문을 현시한다.

본문현시는 76행에서 시작하는 메쏘드 updateDisplay()에 의해 수행된다. 본문을 현시하기 위하여서는 81행에서 QPainter객체를 만들어야 한다. QPainter객체는 한가지 서체를 포함하며 그것을 리용하여 그 본문을 모두 그린다. 82행에서 선택된 서체는 QPainter객체에 설정된다. 83행에서 이 서체에 대한 정보를 포함하는 QFontMetrics객체는 QPainter객체로부터 얻어진다.

알아두기: QPainter클라스는 12장에서 설명한것처럼 많은 기본도형조작함수들에서 사용된다. 이 실례에서 QPainter구성자는 틀을 목표창문부품으로 사용하지만 그것은 또한 창문부품에 독립인 QPainter객체를 만들수 있으며 오직 그리기를 수행해야 할 때만 련결될수 있다.

85행은 배경색을 흑색으로, 전경색을 흰색으로 설정한다. 이것은 창문부품이 현시될 때 그것이 흑색으로 보이고 거기에 그린 본문은 백색으로 보인다는것을 의미한다.

본문의 수직 및 수평위치에 대한 3가지 선택이 있는데 이것들은 91행과 106행에서 switch문에 의해 선택된다. 문자렬을 그리기 위하여서는 기준선의 수직위치와 왼쪽문자의 왼끝의 수평위치를 지정하여야 한다. 서체측도정보를 사용하여 문자렬을 배치하는데 필요한 정확한 위치를 결정할수 있다.

93행에서 y를 서체의 상승값으로 설정하여 본문을 우에 배치한다. 즉 기준선의 수직배치는 가장 큰 문자의 웃끝이 바로 창문의 웃끝에 닿아야 한다.

97～98행에서 본문이 중심에 나타나도록 수직위치를 설정한다. 97행의 식은 창문의 수직중심을 결정하지만 상승과 하강이 각이한 값을 가지는것이 거의 확실하므로 본문(기준선이 아니다)이 중심에 배치되도록 조절해야 한다. 98행은 상승과 하강사이의 차이를 구하여 중심에 그 차를 추가한다. 그것을 2개 명령문으로 나누지 않고 다음과 같이 쓸수 있다.

```
y = ( height + ascent - descent ) / 2;
```

102행은 이미 그림 10-8에서 보여준것처럼 최소서체하강이 창문의 아래에 놓이도록 수직상태를 계산한다. 그러기 위하여 창문의 전체 높이에서 하강을 덜어야 한다.

108행은 창문의 왼쪽에서 본문을 시작한다. 문자렬은 항상 그 x자리표의 바로 오른쪽에 그려지므로 x를 0으로 설정할 필요만 있다.

112행은 본문이 수평으로 중심에 배치되도록 x자리표를 결정한다. 창문너비의 절반은 창문의 중심이다. 왼쪽의 위치를 문자렬길이의 절반만큼 조절하면 문자렬이 정확히 중심에 배치된다. width메쏘드호출은 인수로서 본문을 사용한다. 이것은 문자렬의 문자수로부터가 아니라 매개 실제 문자너비의 합계로부터 계산되기때문이다. Courier와 같은 고정너비서체들은 간단히 문자수로부터 너비를 계산할수 있으나 가변너비서체들은 한번에 한 문자씩 계산해야 한다.

117행은 그림 10-8에서 이미 보여준것처럼 본문의 마지막 문자가 창문의 오른쪽에 맞추어 끝나도록 본문의 출발점을 계산한다. 전체 창문의 너비에서 문자렬너비를 덜어 출발점을 계산한다. 일단 본문문자렬을 구성하고 x와 y자리표들을 계산한 다음에 120행의 drawText() 호출은 창문에 본문을 그리는데 사용된다.

122행의 메쏘드 paintEvent()는 창문이 어떠한 리유로 로출되여 다시 그려야 할 때마다 호출된다. 이 메쏘드가 여기서 정의되지 않았다면 창문은 오직 새로운 서체 또는 새로운 위치가 선택될 때에만 그려진다.

제7절. 직4각형에 의한 서체배치

이전의 실례에서는 QFontMetric객체를 사용하여 직4각형안에서 여러가지 서체위치를 계산하였다. updateDisplay()메쏘드를 다음 코드로 바꾸면 앞에서와 꼭같이 현시된다.

```
…
76 void FontPaint2::updateDisplay()
77 {
78     int align;
79     QString text;
80     QPainter painter(frame);
81     painter.setFont(font);
82
83     painter.setBackgroundColor(QColor("black"));
84     painter.setPen(QColor("white"));
85
86     QRect rect = painter.window();
87     painter.eraseRect(rect);
88
89     switch(Vposition) {
90         case Vtop:
91             align = AlignTop;
92             text = "Top ";
93             break;
94         case Vmiddle:
95             align = AlignVCenter;
96             text = "Middle ";
97             break;
98         case Vbottom:
99             align = AlignBottom;
100            text = "Bottom ";
101            break;
102    }
```

```
103  switch(Hposition) {
104        case Hleft:
105              align |= AlignLeft;
106              text += "Left";
107              break;
108        case Hcenter:
109              align |= AlignHCenter;
110              text += "Center";
111              break;
112        case Hright:
113              align |= AlignRight;
114              text += "Right";
115              break;
116  }
117  painter.drawText(rect,align,text);
118 }
...
```

이 실례에서 align변수는 기발들의 모임으로서 사용된다. 117행의 drawText()호출은 직4각형안의 본문위치를 지정하는 기발들과 함께 전체 창문의 크기를 정의하는 QRect객체를 사용한다. 모든 기발들에 대하여서는 표 10-2에서 설명한다. 이전 실례에서 사용한것처럼 본문위치를 지정하는 실제 화소값들은 필요되지 않는다. 또한 직4각형은 전체 창문을 포함할 필요가 없고 창문내부에 더 작은 직4각형을 지정할수 있다.

표 10-2. 직4각형안에 넣어진 본문의 위치를 지정하는 기발

기발이름	작 용
AlignBottom	서체의 최소하강위치의 꼭대기가 직4각형의 바닥에 일치하도록 본문위치를 지정한다. AlignTop 또는 AlignVCenter와 함께 사용할수 없다.
AlignHCenter	본문을 수평방향으로 중심에 배치한다. AlignLeft 또는 AlignRight와 함께 사용할수 없다.
AlignLeft	본문의 제일 왼쪽 문자는 직4각형의 왼변과 일치한다. AlignRight 또는 AlignHCenter와 함께 사용할수 없다.
AlignRight	본문의 제일 오른쪽 문자는 직4각형의 오른변과 일치한다. AlignLeft 또는 AlignHCenter와 함께 사용할수 없다.
AlignTop	서체에서 제일 큰 문자의 꼭대기가 직4각형의 웃변과 일치하도록 본문위치를 지정한다. AlignBottom 또는 AlignVCenter과 함께 사용할수 없다.
AlignVCenter	직4각형의 수직방향으로 중심에 본문을 배치한다. AlignTop 또는

기발이름	작 용
	AlignBottom과 함께 사용할수 없다.
DontClip	만일 직4각형이 창문과 본문보다 작으면 문자렬은 직4각형에 맞게 잘리운다. 이 기발을 리용하면 본문은 직4각형에 맞추어 잘리우지 않는다.
ExpandTabs	기정으로 매개 타브문자 '\t'는 하나의 공백으로 바뀐다. 이 기발을 리용하면 다음 문자가 문자렬의 선두로부터 계산하여 8문자경계에서 표시되도록 충분한 공백을 삽입한다.
ShowPrefix	기정으로 본문의 &문자는 그대로 나타난다. 이 기발을 리용하면 &문자는 삭제되고 그 오른쪽문자에 밑줄이 그어진다. 실례로 본문 "Mi&ddle &Center"는 이 기발을 리용하면 Mi<u>d</u>dle <u>C</u>enter로 나타나며 이 기발을 리용하지 않으면 Mi&ddle &Center로서 나타난다.
SingleLine	기정으로 새행문자 '\n'는 본문을 분리하여 한행이상에 표시되게 한다. 이 기발을 사용하면 매개 newline문자는 하나의 공백으로 전환된다.
WordBreak	본문이 직4각형에 맞지 않으면 이 기발은 그것이 맞도록 하기 위하여 공백위치에 새행문자를 삽입하게 한다.

본문은 여러행으로 분리될수 있으며 그 결과로 생기는 블로크는 alignment기발들에 따라서 적합한 위치에 배치된다. 앞의 실례에서 본문의 두 단어는 하나의 공백으로 식별되였으므로 한행에 나타났다. 그림 10-9에서는 이전 실례의 코드를 다음과 같이 바꿈으로써 단어들사이에 새행문자 '\n'를 사용하였을 때 발생하는 배치를 보여준다.

```
...
89  switch(Vposition) {
90          case Vtop:
91                  align = AlignTop;
92                  text = "Top\n";
93                  break;
94          case Vmiddle:
95                  align = AlignVCenter;
96                  text = "Middle\n";
97                  break;
98          case Vbottom:
99                  align = AlignBottom;
100                 text = "Bottom\n";
101                 break;
102 }
```

…

그림 10-9. 창문에 여러행본문의 배치

요 약

도형방식사용자대면부에 현시되는 모든 요소들중에서 기본은 문자, 수, 그리고 구두점찍기이다.

몇개의 문자서체들은 매우 매력있어 보이며 반면에 다른 문자서체들은 나쁘게 보인다. 우리가 좋아하는 서체와 좋아하지 않는 서체사이의 실제차이는 매우 작다. 문자들이 보통 매우 작은 도형객체로 창문에 현시되여도 우리는 그 상세한 모양을 곧 인식할수 있다. 문자모양의 이러한 예상된 표기는 서체가 우리에게 아주 훌륭하게 보일수 있기때문이다.

이 장에서는 서체조작의 기초를 설명하였다. 즉

- 매개 서체는 그 자체의 파일에 보관되므로 새로운 서체들을 추가하거나 이전 서체를 쉽게 삭제할수 있다.
- 자기 응용프로그람은 정확한 이름으로 특정한 서체를 서술할수 있다. 또는 서체계렬과 점크기와 같은 서술적인 용어를 사용함으로써 서체를 선택할수 있다.
- 창문에서 표준값모임을 리용하여 본문위치를 지정하고 현시하는 본문의 크기를 결정할수 있다.
- 사용자가 서체를 선택할수 있게 하기 위하여 QFontDialog 또는 KFontDialog창문부품을 프로그람에 포함할수 있다.

이 장에서는 창문부품의 창문에 본문을 그리는 방법을 론의하였다.

제11장. 색

학습내용

체계가 색을 만드는 방법

단일색을 포함하는 객체의 만들기

사용자에게 색선택을 의뢰

색들을 련관된 그룹으로 묶기

색그룹들을 조색판으로 묶기

한개 창문부품, 여러 창문부품 혹은 모든 창문부품용으로 조색판을 지정하는 방법

이 장에서는 응용프로그람의 대면부를 만드는 여러가지 창문부품들의 색을 설정하는 방법을 설명한다. 모든 창문부품들에서 사용하는 모든 색을 표준화하기 위한 채색기술을 사용하거나 개별적인 창문부품에 유일한 색을 지정할수 있으며 또는 이 두가지 수법을 결합하여 사용할수 있다. KDE와 Qt는 사용자의 요구에 따라 창문부품색을 설정 및 재설정할수 있다.

모든 X11색은 그 기본적인 원색부분들로 분리할수 있는 하나의 수값으로서 정의된다. QColor객체는 하나의 X11색을 보관하는 용기이다. QColorGroup객체는 창문부품창문의 여러가지 부분을 착색하는데 사용된 한조의 QColor객체를 가진다. QPalette객체는 3조의 QColorGroup를 포함하는데 그것은 창문부품의 매개 상태에 대한 색을 지정하는데 사용된다. 이것들은 응용프로그람의 조종하에 있다.

가장 낮은 준위에서 색은 현시된 화소에 적용된 수값이다. 기본원칙은 같지만 도형기판에 따라서 약간 다른 수법들을 리용한다. 이 장은 주로 그 수법들을 설명한다. 일부 도형기판은 자기의 체계에서 색을 현시할수 있는 수법들을 자체로 인식하지만 기판은 사용자들이 구동프로그람을 적재하여야 한다.

제1절. 색의 구조

X창문체계의 색체계는 2진값으로 표시된 3가지 원색을 사용한다. 매개 색의 세기는 가능한 최대값에 대한 색값의 비률이다. 실례로 색당 8bit를 가지는 체계에서 50% 적색을 얻으려고 할 때 값은 127이다. 색당 16bit를 가지는 체계에서 50%적색을 얻으려고 할 때에 값은 32,767이다. 또한 색은 대체로 0.0～1.0범위의 류동소수점값으로 표시될수 있으므로 50%색준위는 0.5이다.

현시구조의 종류는 아주 많다. X11은 표준방법으로 그것들을 다루는 방법을 제안하였다. KDE/Qt쏘프트웨어는 이와 같이 일반화된 체계우에 구축되므로 저준위의 세부를 모두 알 필요는 없다. 그러나 프로그람이 말아 수행하여야 할 일부 조작들을 리해하려면 일정한 기초가 있어야 한다.

물리적인 현시장치는 거기에 배치하는 매개 화소에 대한 위치를 기억하고 하드웨어기억기를 가지고있다. 기억기의 매개 값은 그와 련결된 화소의 색과 밝기를 결정하며 화소를 변경하려면 그 기억장소의 내용을 변경하여야 한다. 하드웨어는 기억된 값을 색으로 변환할 때 다른 방법을 사용한다. 즉 기억기의 일부 수값은 직접 색값으로 바꾸며 나머지 수값은 색표에 대한 색인으로 사용한다. 색표는 색략도로 알려져있다. 하드웨어에 따라 각이한 종류의 색략도를 요구한다. 표 11-1은 여러가지 형들을 보여준다.

표 11-1. 물리현시장치의 부류

이름	설명
가상색 (Pseudo Color)	화소값은 RGB값들을 포함하는 색략도를 색인한다. 색략도는 동적으로 변경될수 있다.
직접색	separate화소값은 3개 값으로 구분되고 3개의 독립적인 색략도를 색인하는데 쓰인다. 즉 적색요소에 하나, 청색요소에 하나, 록색요소에 하나. 색략도들은 동적으로 변경될수 있다.
회색비례 (Gray Scale)	화소값은 현시가능한 회색비례(scale)값을 포함하고있는 색략도를 색인한다. 색략도는 동적으로 변경할수 있다.
정적색	화소값은 RGB값을 포함하는 색략도를 색인한다. 색략도는 하드웨어에서 정적이고 변경될수 없다.
참색 (True Color)	화소값은 3개의 값으로 분리되고 3개의 독립적인 색략도를 색인하는데 쓰인다. 즉 적색요소에 하나, 푸른색요소에 하나, 그리고 록색요소에 하나. 매개 색략도는 무색으로부터 완전포화까지 균일한(또는 근사균일한) 경사도이며 변경할수 없다.
정적회색	화소값은 현시가능회색비례값들을 포함하는 색략도를 색인한다. 색략도는 정적이고 변경될수 없다.

색략도는 모든 창문의 모든 화소들을 그리기 위하여 현시기하드웨어에 의해 사용되므로 색략도의 변경은 모든 모양을 변경한다. 일부 색략도는 이러한 변경을 허용하며 일부는 허용하지 않는다. 드물게 색략도를 변경할 필요가 있지만 응용프로그람이 색략도를 변경하면 다른 응용프로그람으로 옮겨 변경하고 다시 응용프로그람으로 되돌아와야 한다.

다음 프로그람은 자기가 어떤 종류의 현시기를 가지고있으며 화소당 비트수, 색략도의 크기 그리고 일부 다른 관련정보를 현시한다.

```
1 /* showvisual.cpp */
2 #include <kapplication.h>
3 #include <qlabel.h>
4 #include <qlayout.h>
```

```
5 #include <X11/Xlib.h>
6 #include "showvisual.h"
7
8 int main(int argc, char **argv)
9 {
10    KApplication app(argc, argv,  "showvisual");
11    ShowVisual showvisual;
12    showvisual.show();
13    app.setMainWidget(&showvisual);
14    return(app.exec());
15 }
16 ShowVisual::ShowVisual(QWidget *parent,const char *name)
17    : QWidget(parent,name)
18 {
19    QString str;
20    QLabel *label;
21
22    QVBoxLayout *vbox = new QVBoxLayout(this,10);
23
24    str.sprintf("%4d Screen number",x11Screen());
25    label = new QLabel(str,this);
26    vbox->addWidget(label);
27
28    str.sprintf("%4d Bits per pixel",x11Depth());
29    label = new QLabel(str,this);
30    vbox->addWidget(label);
31
32    str.sprintf("%4d X dots per inch",x11AppDpiX());
33    label = new QLabel(str,this);
34    vbox->addWidget(label);
35
36    str.sprintf("%4d Y dots per inch",x11AppDpiY());
37    label = new QLabel(str,this);
38    vbox->addWidget(label);
39
```

```
40	Visual *visual = (Visual *)x11Visual();
41
42	str.sprintf("%4d Bits per RGB",
43		visual->bits_per_rgb);
44	label = new QLabel(str,this);
45	vbox->addWidget(label);
46
47	str.sprintf("%4d Colormap entries",
48		visual->map_entries);
49	label = new QLabel(str,this);
50	vbox->addWidget(label);
51
52	switch(visual->c_class) {
53		case StaticGray:
54			str.sprintf("%4d StaticGray class",
55				visual->c_class);
56			break;
57		case GrayScale:
58			str.sprintf("%4d GrayScale class",
59				visual->c_class);
60			break;
61		case StaticColor:
62			str.sprintf("%4d StaticColor class",
63				visual->c_class);
64			break;
65		case PseudoColor:
66			str.sprintf("%4d PseudoColor class",
67				visual->c_class);
68			break;
69		case TrueColor:
70			str.sprintf("%4d TrueColor class",
71				visual->c_class);
72			break;
73		case DirectColor:
74			str.sprintf("%4d DirectColor class",
```

```
75                visual->c_class);
76            break;
77    }
78    label = new QLabel(str,this);
79    vbox->addWidget(label);
80
81    str.sprintf("0x%08X Red Mask",
82        visual->red_mask);
83    label = new QLabel(str,this);
84    vbox->addWidget(label);
85
86    str.sprintf("0x%08X Green Mask",
87        visual->green_mask);
88    label = new QLabel(str,this);
89    vbox->addWidget(label);
90
91    str.sprintf("0x%08X Blue Mask",
92    visual->blue_mask);
93    label = new QLabel(str,this);
94    vbox->addWidget(label);
95
96    resize(10,10);
97 }
```

이 프로그람에 의해 현시된 모든 정보는 실제로 저준위 X11체계에 의해 제공된다.

QWidget객체는 QPaintDevice기초클라스로부터 모든 메쏘드들을 계승하므로 같은 정보를 어떠한 창문부품으로부터 검색할수 있다. 그림 11-1은 동시에 256색을 현시할수 있는 체계가 생성한 정보를 현시한 색특성을 보여준다.
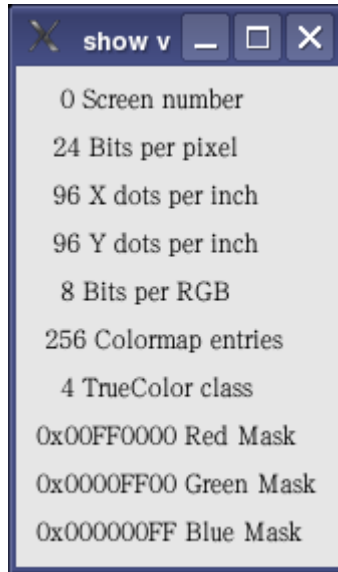
그림 11-1. 현시의 색특성

24～26행은 화면번호를 현시하는 표식자를 만든다. 이 번호는 항상 0이지만 X11이 많은 화면을 동시에 조종하도록 설계되므로 매 화면이 서로 다른 번호를 가질수 있다. 또한 X11은 망용으로 설계되였으므로 어떤 화면들은 다른 콤퓨터표시장치에 현시할수 있다. 즉 같은 응용프로그람이 여러 콤퓨터들에 화면을 동시에 현시하게 할수 있다.

28～30행은 매개 화소에 할당된 하드웨어비트수를 현시하는 표식자를 만든다. 같은 값을 가지는 2개 화소는 같은 색을 정확히 가지므로 총동시색수는 256으로 제한된다.

32～38행은 수평과 수직 두 방향에서 인치당 점의 개수를 지정하는 한쌍의 표식자를 만든다. terminal의 환경구성에 대하여 쏘프트웨어가 아는 방법이 없으므로 이 값은 오직 근사값이다. CRT는 조절할수 있는 너비와 높이를 가지고 있으며 현시구역에서 모든 변들에서 직선으로 되는것은 아니다.

40행의 x11Visual()호출로 Visual구조체를 얻으며 이 구조체는 현시기에 대한 기본정보를 가진 저준위X11구조체이다. 이 값들은 현시된 창문의 나머지 부분을 만드는데 사용된다. 이 구조체는 X쏘프트웨어의 부분이므로 5행에서 수행된것처럼 파일 Xlib.h를 포함하여야 한다.

42～45행은 RGB값당 6bit가 있다는것을 알리는 표식자를 만든다. 즉 18bit(매개 원색에 6)는 완전색값을 표시한다. 색의 투명도를 지정하는 다른 6bit값(알파값)이 흔히 있으므로 RGB당6화소는 자주 24bit색체계로서 언급된다.

매개 6bit옹근수들이 0～63의 값을 포함할수 있고 64의 3제곱이 262144이므로 그것은 이 체계가 보관할수 있는 색의 총수이다. 그러나 그것은 동시에 256색을 현시할수 있다. 47～50행은 색략도안의 총항목수를 보여주는 표식자를 만든다. 수 256은 화소당 8 bit이므로 당연한 값이다. 색략도배렬의 매개 성원은 24bit이므로 각자는 색의 4개 값(적색, 록색, 청색 그리고 알파)을 가질수 있다.

52～76행은 현시기의 부류를 지정하는 표식자를 만든다. 형이름은 5행에서 포함되는 X머리부파일들에서 선언된다. 이 실례에서 8bit화소에 보관한 값이 색략도배렬의 색인으로 사용된다는것을 의미하는 가상색을 현시기가 리용한다. 그리고 색략도의 값들은 응용프로그람이 조절할수 있다.

81～94행은 3가지 색마스크의 값들을 현시하는 표식자들을 만든다. 그것들은 오직 참색과 직접색에만 적용하므로 이 실례에서 0이다. 이 현시기부류들은 둘다 매개 원색에 대하여 독립적인 색략도를 사용하며 마스크들은 화소값들로부터 매개 배렬에로 색인값들을 얻어내는데 사용된다. 사실상 매개 색략도가 오직 한가지 색을 포함하고 심지어 무색(흑색)으로부터 완전한 색밝기까지 그림자처리가 있으므로 흔히 실제색략도는 없다. 대신에 마스크의 크기는 색마다 다를수 있으며 색이 이 두 한계값들사이의 어느 값을 가지는가를 결정하는데 사용된다.

X11체계로부터 창문과 화면에 대한 정보가 더 있으나 아마도 그것들을 구체적으로 알 필요는 없다. X11루틴들을 피하는것의 우점들중 하나는 자기의 코드가 더 간단해지는것이다. 게다가 자기가 작성한 응용프로그람은 한 체계에서 다른 체계에로 이식할수 있다.

제2절. QColor객체의 구성

QColor객체는 색의 정의를 포함한다. QColor객체를 만드는 방법은 많다. 다음 프로그람은 그림 11-2에 보여준 도색된 표식자모임을 창조함으로써 QColor객체를 만드는 서로 다른 방법을 보여주는데 매번 QColor구성자인수의 각이한 모임을 사용한다.

```
1 /* colormaker.cpp */
2 #include <kapplication.h>
3 #include <qlabel.h>
4 #include <qlayout.h>
5 #include "colormaker.h"
6
7 int main(int argc, char **argv)
8 {
9     KApplication app(argc, argv,  "colormaker");
10    ColorMaker colormaker;
11    colormaker.show();
12    app.setMainWidget(&colormaker);
13    return(app.exec());
14 }
15 ColorMaker::ColorMaker(QWidget *parent,const char *name)
16    : QWidget(parent,name)
```

```
17 {
18     QString str;
19     QLabel *label;
20
21     QVBoxLayout *vbox = new QVBoxLayout(this,3);
22
23     label = new QLabel("Defined by RGB numbers",this);
24     label->setBackgroundColor(QColor(250,150,100));
25     vbox->addWidget(label);
26
27     label = new QLabel("Defined by RGB numbers",this);
28     label->setBackgroundColor(QColor(150,250,100,
29             QColor::Rgb));
30     vbox->addWidget(label);
31
32     label = new QLabel("Defined by HSV numbers",  this);
33     label->setBackgroundColor(QColor(310,150,250,
34             QColor::Hsv));
35     vbox->addWidget(label);
36
37     label = new QLabel("Defined by QRgb value",  this);
38     QRgb rgb = 0x00F0E000;
39     label->setBackgroundColor(QColor(rgb));
40     vbox->addWidget(label);
41
42     label = new QLabel("Defined by colormap index",this);
43     label->setBackgroundColor(QColor(rgb,86));
44     vbox->addWidget(label);
45
46     label = new QLabel("Defined by RGB name",this);
47     label->setBackgroundColor(QColor("#F58F95"));
48     vbox->addWidget(label);
49
50     label = new QLabel("Defined by file name",this);
51     label->setBackgroundColor(QColor("green"));
```

```
52    vbox->addWidget(label);
53
54    resize(10,10);
55 }
```
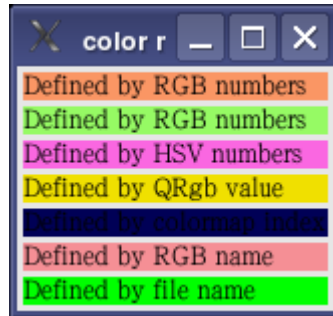
그림 11-2. 색을 만드는 몇가지 기본방법

매개 창문부품은 배경색을 가지고있다. 이 실례는 QLabel창문부품집합의 배경색을 QColor객체를 사용하여 설정한다. 배경색은 창문부품의 배경을 색칠하는데 사용된다. QLabel에서 배경색으로 전체 창문부품을 채색한 다음 본문을 그 우에 쓴다. QColor객체를 만들 때마다 구성자가 현재 색략도에 새로운 색의 삽입이 필요할수 있다. 색이 색략도에 이미 있다면 어떤 작용도 필요하지 않다. 그러나 색을 추가해야 하는데 색략도에 빈자리가 없으면 QColor객체는 색략도에서 가장 가까운 위치에 있는 색에 간단히 련결한다. 이것때문에 많은 색을 가지는 응용프로그람은 상황에 따라 좀 다르게 보일수 있다.

23～25행은 RGB값으로 지정된 색을 가지는 표식자를 만든다. 매개 값은 1～255범위에 있으며 여기서 0은 색이 없고 255는 최대량을 가진다. RGB값 0, 0, 0은 흑색이고 255, 255, 255값은 백색이다.

27～30행은 3개 값이 RGB값으로 해석된다는것을 구성자에 알리는 추가적인 인수를 내놓으면 이전 실례에서와 같이 RGB값을 사용하여 색을 만든다.

32～35행은 28행에서와 같이 QColor구성자를 사용하지만 RGB대신에 HSV(색, 농도, 값)를 사용하여 QColor객체를 만든다. RGB색에서처럼 HSV색도 3개 인수에 의해서 정의되지만 수값들은 아주 다른 의미를 가지고있다. 또한 HSB(색, 농도, 밝기)가 HSV와 련관된다는것을 알수 있다. H값은 색(hue, tint)으로서 색스펙트르에서 빛의 빈도수를 지정한다. S값은 농도(saturation, shade)로서 기준색이 더 밝아지거나 어두워지도록 혼합하는 흑색이나 백색의 량을 지정한다. B 또는 V값(밝기brightness 또는 빛세기luminosity)는 색이 현시되는 농도를 지정한다.

HSV색을 만들 때 구성자에 넘긴 H값은 색을 선택하기 위하여 0～360범위에 있거나 또는 -1로서 색(회색, 흑색 또는 백색)이 없다는것을 지정할수 있다. S값은 0(최대로 흑색이 포함)～255(최대로 백색이 포함)에서 변한다. 순수색을 얻기 위하여 농도로서 127을 사용한다. V 또는 B값은 0(최소밝기)～255(최대밝기)에서 변한다.

37～40행은 RGB값으로 색을 지정하지만 3개 값은 모두 한개 옹근수에 보관된다. 편리

상 값 그자체는 38행에서 16진값으로 선언된다. 값의 첫째 바이트는 무시되고 적색값은 0xF0, 록색은 0xE0 그리고 청색은 0x00이다.

42～44행은 색략도에 대한 색인을 사용함으로써 색을 지정한다. 구성자의 둘째 인수(수값 86)는 색략도에 대한 색인이고 첫째 인수는 RGB값에 포함되는 옹근수이다. 색인값은 부호없는 수이지만 0xFFFFFFFF로서 그것을 지정하려고 하였다면 색인은 무시되고 RGB값이 대신에 사용된다.

46～48행은 색을 지정하기 위하여 RGB값의 16진문자형식을 사용한다. 실례에 사용한 수값문자렬형식은 "#RRGGBB"이지만 그것은 "#RGB", "#RRRGGGBBB" 또는 "#RRRRGGGGBBBB"일수도 있다. 색정의쏘프트웨어는 선두의 #문자를 탐지하고 값들을 꺼내기전에 나머지 문자렬을 같은 부분들로 나눈다.

50～52행은 파일 /usr/lib/X11/rgb.txt의 항목으로부터 만들어진 QColor객체를 사용한다. 이것은 평문파일이고 매개 항목은 색에 대한 이름과 RGB값을 포함한다. 실례로

60 179 113 MediumSeaGreen

32 178 170 LightSeaGreen

152 251 152 PaleGreen

0 255 0 green

0 250 154 MediumSpringGreen

이 파일에 750개 항목이 있고 파일은 X11과 함께 배포되므로 거기에 있는 이름들을 사용하는것이 안전하다고 느낄수 있다. 거의 모든 색이 하나이상의 이름을 가지고 있으므로 파일에 750이하의 유일한 색이 있다. 또한 원한다면 자기의 이름을 추가하고 자기의 프로그람이 그것들을 리용할수 있으나 자기 응용프로그람을 이식할수 없다.

제3절. KColorDialog

응용프로그람에 색을 설정하는 수법은 여러가지 있다. 간단히 색을 무시하고 기정값들을 사용할수 있으며 매개 창문부품에 특정한 색값을 설정하거나 사용자가 색을 선택하게 할수 있다. KColorDialog는 사용자가 색을 선택할수 있는 튀여나오기대화칸이다. 다음 실례는 KColorDialog를 펼치고 사용자가 선택한 색정보를 얻는 방법을 보여준다.

그림 11-3에 보여주는 응용프로그람의 기본창문은 색현시대화칸을 펼치는 단추를 아래에 포함한다. 창문의 꼭대기에 현재 선택된 색의 16진RGB(적, 록, 청)값이 있다. 창문의 중심에 색을 현시하는 블로크가 있다.
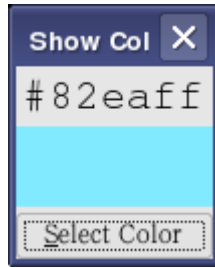
그림 11-3. 색현시대화칸

ShowColor머리부파일

```
1 /* showcolor.h */
2 #ifndef SHOWCOLOR_H
3 #define SHOWCOLOR_H
4
5 #include <qwidget.h>
6 #include <qlabel.h>
7 #include <qstring.h>
8
9 class ShowColor: public QWidget
10 {
11     Q_OBJECT
12 public:
13     ShowColor(QWidget *parent=0,const char *name=0);
14 private:
15     QLabel *label;
16     QWidget *widget;
17     QColor color;
18     QString colorName;
19 private slots:
20     void popup();
21 };
22
23 #endif
```

ShowColor클라스는 그림 11-3에 기본창문을 현시하는 창문부품이다. 그것은 RGB값을 표시하는 표식자와 중간에 색을 현시하는 창문부품을 포함한다. 또한 QColor객체에 현재색을 보관하고 색이름을 QString객체에 보관한다. 색이름은 RGB값의 16진표시이다. 처리부메쏘드 popup()은 단추를 찰칵할 때마다 KColorDialog를 현시하는데 사용된다.

ShowColor

```
1 /* showcolor.cpp */
2 #include <kapplication.h>
3 #include <qpushbutton.h>
4 #include <qlayout.h>
5 #include <kcmdlineargs.h>
6 #include <kcolordlg.h>
7 #include "showcolor.h"
8
9 int main(int argc, char **argv)
10 {
11     KCmdLineArgs::init(argc, argv,  "showcolor",
12             "Show Color","0.0");
13     KApplication app;
14     ShowColor showcolor;
15     showcolor.show();
16     app.setMainWidget(&showcolor);
17     return  (app.exec());
18 }
19 ShowColor::ShowColor(QWidget *parent,const char *name)
20     : QWidget(parent,name)
21 {
22     QVBoxLayout *box = new QVBoxLayout(this,0,3);
23
24     colorName = "#FF0000";
25     color.setNamedColor(colorName);
26     label = new QLabel(colorName,this);
27     label->setFont(QFont("Courier",16));
28     label->setAlignment(Qt::AlignHCenter);
29     box->addWidget(label);
30
31     widget = new QWidget(this);
32     widget->setFixedHeight(40);
33     widget->setBackgroundColor(color);
34     box->addWidget(widget);
```

```
35
36    QPushButton *button = new QPushButton("Select Color",
37            this);
38    box->addWidget(button);
39    connect(button,SIGNAL(clicked()),
40            this,SLOT(popup()));
41
42    resize(10,10);
43    box->activate();
44 }
45 void ShowColor::popup()
46 {
47    int cond = KColorDialog::getColor(color,this);
48    if(cond == KColorDialog::Accepted) {
49            colorName = color.name();
50            label->setText(colorName);
51            widget->setBackgroundColor(color);
52    }
53 }
```

편리상 이 원천파일은 main()함수와 ShowColor클라스의 실행가능코드를 모두 포함한다. 9행에서 시작하는 main()함수는 간단히 최상위창문을 만들고 그안에 ShowColor창문부품을 삽입한다.

19행에서 시작하는 ShowColor구성자는 수직칸을 만들고 거기에 3개 창문부품을 삽입한다. 24～25행은 초기색을 적색으로 설정한다. 이름을 현시하는 표식자는 26행에서 만들어진다. 27행은 setFont()를 호출하여 고정서체로 16진수들을 현시한다. setAlignment()호출은 중심에 본문을 배치하고 29행의 addWidget()호출은 칸에 표식자를 삽입한다.

창문의 중심에 현시된 색블로크는 간단히 배경색만 설정된 창문부품이다. 그것은 31행에서 창조되며 그 높이는 28행에서 40화소로 고정된다. 수직칸이 그 너비를 조종하므로 그것을 설정할 필요는 없다. 33행에서 setBackgroundColor()가 호출되여 다른 방법으로 빈 창문부품을 칠한다.

36～40행은 단추를 만들어 칸에 삽입하고 popup()에 련결한다.

45행에서 시작하는 popup()은 단추를 누를 때마다 실행된다. 그것은 그림 11-4에 보여주는 KColorDialog창문을 현시하는 정적메쏘드 getColor()호출에 의해 시작된다. 이 메쏘드는 사용자가 색을 선택하거나 선택하지 않은채 대화칸을 닫을 때까지 돌아오지 않는다. 색이 선택되였으면 cond의 값은 색이 선택되였다는것을 가리키는 상수값 KColorDialog::Accepted과

갈아진다. 색이 선택되면 그것은 getColor()에 넘긴 첫째 인수로서 QColor객체에 놓인다. 49행에서 새로운 색의 이름을 얻고 50행에서 이름이 표식자에 삽입된다. 51행은 setBackgroundColor()를 호출하여 창문부품에 의해 중심에 현시되는 색을 갱신한다.
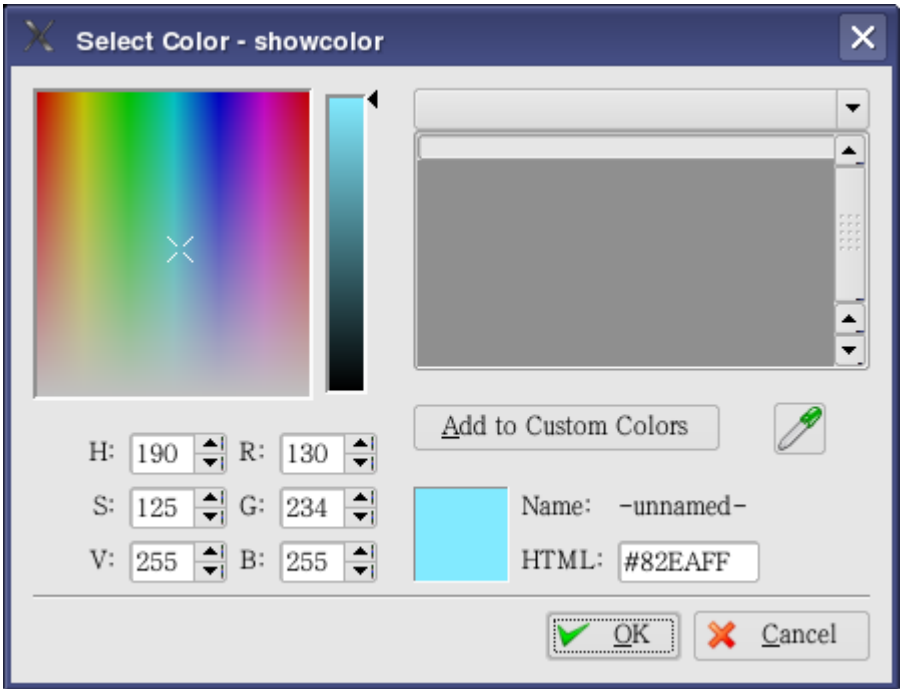
그림 11-4. KColorDialog창문

47행의 getColor()호출은 첫째 인수로서 QColor객체를 사용한다. 넘긴 색은 기정색값이고 그것은 KColorDialog창문의 현시를 초기화한다. 이 실례는 늘 현시되는 색을 넘기므로 대화칸의 기정색은 현재색이다. KColorDialog창문부품은 여러가지 방법으로 색을 선택하는데 사용된다. 왼쪽웃구석의 그룹은 체계색의 집합을 제시하는데 체계색은 KDE체계의 부분으로 정의되여 대부분의 구성요소들에서 사용된다. 이 색들은 각이한 콤퓨터들에서 대체로 동일하다. 또한 색은 오른쪽웃구석의 2개 색선택칸에서 마우스를 사용하여 선택할수 있다. 큰 칸의 둘레에 +지시자를 끌고갈 때 오른쪽 아래에서 RGB와 HSV값들이 변화하는것을 보게 된다. 색을 선택하는 세번째 방법은 왼쪽 아래에서 집합에 보관한 사용자정의색들중 하나를 선택하는것이다.

알아두기: 매개 사용자가 사용하는 사용자정의색은 파일 ~/kde/share/config/kdeglobals에 보관되며 KColorDialog가 펼쳐질 때마다 얻어진다.

사용자정의색을 추가하려면 우선 사용자정의색들의 집합에서 한개 칸을 선택한다. 다음에 왼쪽의 체계색으로부터 또는 오른쪽의 색판으로부터 마우스로 색을 선택한다. 일단 선택한 다음에는 Add to Custom Colors표식자가 있는 단추를 눌러서 그것을 삽입한다.

제4절. QColorGroup안의 QColor들

몇가지 색들은 창문부품을 그릴 때 포함할수 있다. 실례로 누름단추는 배경색, 웃그림자색, 바닥그림자색 그리고 본문색을 가지고있다. 단추가 이 모든 색을 그리는데 사용할수 있는 QColor객체가 있어야 한다. 창문부품에 따라 색모임에 대한 각이한 요구가 있다. 요구조건들은 단추를 누르거나 마우스위치가 변할 때 그리고 창문부품우를 지날 때와 같이 짧은 순간에 변할 때이다.

QColorGroup클라스는 창문부품이 하나의 단위에 요구하는 모든 색을 밀봉하도록 설계된다. 각종 창문부품들과 그것들이 가질수 있는 각이한 폼과 함께 QColorGroup은 광범한 색항목을 포함하여야 한다. 표 11-2에서는 매개 QColorGroup에 포함되는 14가지 색을 보여준다.

표 11-2. QColorGroup객체에 포함된 색들

이름	설명
Background	이 색은 거의 모두 창문부품들의 배경에서 사용된다.
Base	이것은 Background로 정의한것보다 밝은 색으로 하려는 창문부품들의 배경색이다. 이것은 흔히 백색이면서 보다 연한 색이다.
BrightText	이 색은 Dark가 배경으로 사용될 때 본문을 표시하는데 쓰일수 있다.
Button	이것은 단추의 배경색이다. 창문부품의 나머지 부분을 이 색으로 도색한다.
ButtonText	이 색은 Button이 배경으로 사용될 때 본문을 표시하는데 쓰인다.
Dark	이 색은 Button색보다 어둡고 단추에 3차원모양을 주기 위한 그림자처리에 Light와 함께 사용된다.
Foreground	이 색은 창문부품의 표면에 문자들을 쓰거나 그리는데 사용된다.
Highlight	이것은 강조되거나 선택된 항목을 칠하는데 사용되는 배경색이다.
HighlightedText	이것은 배경색으로서 Highlight와 대조되는 색으로서 본문을 현시하는데 적합하다.
Light	이 색은 Button색보다 밝고 창문부품에 3차원모양을 주기 위한 그림자처리에 Dark과 함께 사용된다.
Mid	이 색은 복잡한 그림자처리를 요구하는 창문부품들에서 Button과 Dark사이의 값이다.
Midlight	이 색은 복잡한 그림자처리를 요구하는 창문부품들에서 Button과 Light사이의 값이다.
Shadow	이 색은 매우 어둡고 뚜렷한 그림자처리에 사용된다. 그것은 흔히 흑색이다.
Text	이것은 창문부품의 표면에 평본문을 그리는데 사용되는 색이다. 그것은 Foreground와 같다.

다음 프로그람은 창문부품의 현재QColorGroup을 얻고 거기에 포함된 색들을 모두 현시

하기 위한 프로그람이다. 이 응용프로그람은 그룹의 색들을 변경할수 없으므로 기정색과 값들을 현시한다. 그림 11-5에 보여주는것처럼 프로그람은 14가지 색, 색이름 그리고 매개의 16진표시를 모두 현시한다.

```
1 /* showgroup.cpp */
2 #include <kapplication.h>
3 #include <qlabel.h>
4 #include <qpalette.h>
5 #include "showgroup.h"
6
7 struct namelistStruct {
8     QString name;
9     QColorGroup::ColorRole value;
10 } namelist[] = {
11     { "Background", QColorGroup::Background },
12     { "Base", QColorGroup::Base },
13     { "BrightText", QColorGroup::BrightText },
14     { "Button", QColorGroup::Button },
15     { "ButtonText", QColorGroup::ButtonText },
16     { "Dark", QColorGroup::Dark },
17     { "Foreground", QColorGroup::Foreground },
18     { "Highlight", QColorGroup::Highlight },
19     { "HighlightedText", QColorGroup::HighlightedText },
20     { "Light", QColorGroup::Light },
21     { "Mid", QColorGroup::Mid },
22     { "Midlight", QColorGroup::Midlight },
23     { "Shadow", QColorGroup::Shadow },
24     { "Text", QColorGroup::Text }
25 };
26
27 int main(int argc, char **argv)
28 {
29     KApplication app(argc, argv, "showgroup");
30     ShowGroup showgroup;
31     showgroup.show();
32     app.setMainWidget(&showgroup);
33     return (app.exec());
34 }
```

```
35 ShowGroup::ShowGroup(QWidget *parent,const char *name)
36    : QWidget(parent,name)
37 {
38    QHBoxLayout *hbox;
39    QVBoxLayout *vbox = new QVBoxLayout(this,0,3);
40    int size = sizeof(namelist)/sizeof(namelistStruct);
41    for(int i=0; i<size; i++) {
42            hbox = newColorLine(i);
43            vbox->addLayout(hbox);
44    }
45    resize(10,10);
46 }
47 QHBoxLayout *ShowGroup::newColorLine(int i)
48 {
49    QLabel *label;
50    QHBoxLayout *hbox = new QHBoxLayout();
51    QColorGroup group = colorGroup();
52    QColor color = group.color(namelist[i].value);
53
54    label = new QLabel("",this);
55    label->setBackgroundColor(color);
56    label->setMinimumWidth(100);
57    hbox->addWidget(label);
58
59    label = new QLabel(color.name(),this);
60    label->setFixedWidth(60);
61    label->setFont(QFont("Courier"));
62    hbox->addWidget(label);
63
64    label = new QLabel(namelist[i].name,this);
65    hbox->addWidget(label);
66
67    return(hbox);
68 }
```
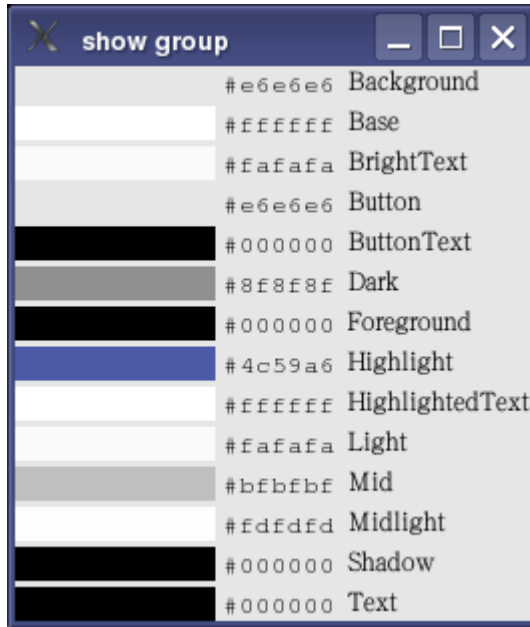
그림 11-5. KColorDialog창문

7∼25행은 매개 색의 이름을 가진 namelist배렬과 QColorGroup으로부터 그것을 얻는데 사용되는 ColorRole값을 선언한다. 배렬은 순환처리를 수행할 때 코드를 간단화하기 위하여 정의되였다.

구성자는 35행에서 시작한다. 최상위용기는 수직칸이다. 41행에서 시작하는 순환고리는 namelist배렬의 매개 요소에 대하여 하나의 수평칸을 구성한다. 43행의 addLayout()호출에 따라 수직칸에 매개 수평칸을 보관함으로써 그림 11-5와 같이 현시한다.

47행에서 시작하는 메쏘드 newColorLine()는 매개 색에 대하여 한번 호출된다. 그때 넘겨진 인수는 namelist배렬의 첨수로 사용된다. 이 메쏘드는 수평칸을 만들고 거기에 3개 표식자를 보관한다. 첫째 표식자는 색자체를 현시하고 둘째 표식자는 색의 16진값, 그리고 셋째 표식자는 색의 QColorGroup이름을 현시한다.

52행은 color()메쏘드를 호출하여 표에서 렬거값을 사용함으로써 QColorGroup로부터 QColor객체를 얻는다. 색을 얻는 다른 방법은 색에 제공된 메쏘드들중의 하나를 호출하는것이다. 실례로 다음 코드행은 Button과 Midlight색을 얻는데 사용될수 있다.

```
QColor bcolor = group.button();
QColor mcolor = group.midlight();
```

54∼57행은 본문없이 표식자를 만들지만 namelist배렬에서 현재 첨수로 설정된 배경색을 가진다. 색은 55행에서 setBackgroundColor()호출에 의해 표식자에 삽입된다.

59∼62행은 색의 16진이름을 보관하는 표식자를 만든다. 이름은 59행에서 name()호출에 의해 QColor객체에서 얻는다. 수값들을 적당히 현시하기 위하여 서체는 고정너비서체인 Courier로 변경하였다.

64행과 65행은 색의 QColorGroup이름을 포함하는 셋째 표식자를 만든다.

제5절. Qpalette의 QColorGroup들

QPalette객체는 3개의 QColorGroup을 보관하는 용기이다. 모든 창문부품은 그것을 가지고있다. 매개 창문부품은 자체를 그릴 때 자기에게 할당된 QPalette객체를 사용한다. 창문부품은 자체를 그릴 때마다 창문부품의 현재 상태에 일치하는 QColorGroup을 사용한다. 3가지 창문부품상태를 표 11-3에서 설명한다.

표 11-3. 창문부품의 3가지 상태

이름	설명
Normal	이것은 기정상태이다.
Active	현재 초점을 가지고있는 창문부품은 Active상태이다. 이 상태의 색들은 보통 Normal상태와 같다.
Disabled	창문부품을 허용하지 않는다. 이 상태의 QColorGroup은 보통 회색이거나 또는 다른 방법으로 창문부품의 동면상태를 가리키도록 한다.

앞의 실례프로그람의 51행에서 QWidget메쏘드 colorGroup()를 호출하여 현재 QColorGroup을 얻는다. 되돌려진 실제의 QColorGroup객체는 창문부품의 현재상태에 의해 결정되므로 3가지중 하나일수 있다. colorGroup()메쏘드사용의 우점은 현재 QColorGroup에서 자기가 요구하는 색을 얻는것이다.

3개의 모든 QColorGroup객체를 창문부품에 사용할수 있다. 다음 코드는 3개의 사본을 얻기 위하여 창문부품이 사용할수 있다.

```
QPalette myPalette = palette();
QColorGroup normalGroup = myPalette.normal();
QColorGroup activeGroup = myPalette.active();
QColorGroup disabledGroup = myPalette.disabled();
```

제6절. 몇가지 창문부품들의 색설정

자체로 작성한 창문부품들의 색에 대한 완전조종뿐아니라 자기 프로그람의 매개 창문부품이 사용하는 모든 색들을 조종하는 창문부품용 색을 정의하는 방법이 있다. 선택적으로 하나의 창문부품 혹은 창문부품나무안의 창문부품들 혹은 나무안에서 선택된 창문부품모임에 대하여 색을 설정하는 방법으로 색을 정의할수 있다.

그것은 자기 창문부품들이 QPalette의 색을 사용하도록 작성하는것이다. 자기의 창문부품을 작성하여 항상 그것이 colorGroup()메쏘드로부터 되돌아온 QColorGroup로부터 색을 얻으면 자기 프로그람도 색에 대한 완전조종을 실행할수 있고 마음대로 그것들을 바꿀수 있다. 사실상 KDE와 Qt에서 제공된 모든 창문부품들은 그 여러 부분을 채색할 때 QWidget의 QPalette에서 제공된 색들을 사용한다.

어떤 창문부품들이 영향을 받는가는 창문부품들을 통하여 조색판환경을 설정하도록 프로그람을 선택하는 방법에 달려있다. 색환경뿐아니라 서체환경에 영향을 주는 선택들을 표 11-4에 보여준다.

표 11-4. 조색판과 서체변경이 자식창문부품들에 주는 영향

이름	설명
NoChildren	이 창문부품의 조색판이나 서체변경은 자식창문부품들에 영향을 주지 않는다.
AllChildren	이 창문부품의 조색판이나 서체변경은 모든 자식창문부품들에도 적용된다.
SamePalette	이 창문부품의 조색판이나 서체변경은 서체 또는 조색판을 설정하지 않은 모든 자식창문부품들에 적용된다.
SameFont	SamePalette와 같다.

특별한 도색을 요구하는 창문부품을 가지고있으나 그 단일창문부품에 색을 적용하려고 한다면 그것이 사용할 QPalette의 특별한 판을 만들수 있다. 42가지 색(3개의 QColorGroup 각각에 14가지 색)을 모두 지정함으로써 모든것을 새로 만들수 있으나 현존 QPalette의 수정판을 창조하려는 경우가 더 많다. 다음 실례는 이것을 수행하는 방법을 보여준다.

```
1 /* colorone.cpp */
2 #include <kapplication.h>
3 #include <qlabel.h>
4 #include <qpushbutton.h>
5 #include <qcolor.h>
6 #include <qlayout.h>
7 #include "colorone.h"
8
9 int main(int argc, char **argv)
10 {
11     KApplication app(argc, argv,  "colorone");
12     ColorOne colorone;
13     colorone.show();
14     app.setMainWidget(&colorone);
15     return  (app.exec());
16 }
17 ColorOne::ColorOne(QWidget *parent,const char *name)
18     : QWidget(parent,name)
```

```
19 {
20    QString str;
21    QLabel *label;
22    QPushButton *button;
23
24    QPalette newPalette = palette().copy();
25
26    QColorGroup normalGroup = newPalette.normal();
27    normalGroup.setColor(QColorGroup::ButtonText,
28            QColor("white"));
29    normalGroup.setColor(QColorGroup::Button,
30            QColor("blue"));
31    normalGroup.setColor(QColorGroup::Foreground,
32            QColor("red"));
33    newPalette.setNormal(normalGroup);
34
35    setPalettePropagation(AllChildren);
36    setPalette(newPalette,TRUE);
37
38    QVBoxLayout *vbox = new QVBoxLayout(this,15);
39
40    button = new QPushButton("The Top Button",this);
41    vbox->addWidget(button);
42
43    label = new QLabel("The Label in the Middle",this);
44    vbox->addWidget(label);
45
46    button = new QPushButton("The Bottom Button",this);
47    vbox->addWidget(button);
48
49    resize(10,10);
50 }
```

24행의 copy()호출은 현재 능동인 QPalette를 복제한다. 새로운 QPalette는 원본의 색값들을 모두 포함하며 원본에 어떠한 영향을 주지 않고 변경할수 있다. copy()호출은 조색판의 사본을 만든다. QPalette객체들이 커질수 있으므로 매개 창문부품은 자기의 비공개사본을 만들

지 않고 사용하는 조색판에로의 참고를 유지한다. 그러므로 palette()메쏘드가 QPalette객체(많은 창문부품들에서 사용되는것일수 있다)의 참고를 돌려주고 자기가 알고있는 그 창문부품들만으로 변경을 제한하려고하므로 copy()메쏘드를 사용하여 그 사본을 작성하는것이 제일 좋다. 이것은 자기의 개별적인 QPalette실례라고 생각할수 있는 사본이다.

QColorGroup은 26행에서 새로운 QPalette로부터 얻는다. 27～32행은 setColor()를 호출하여 색그룹에서 현존색정의중 3개를 교체한다. 이 색들중 2개는 특별히 단추용이고 다른 색은 모든 창문부품들에서 전경색을 변경하기 위하여 설정된다. 33행의 setNormal()호출은 변경된 색그룹을 새로운 QPalette에 보통색그룹으로 보관한다.

35행의 setPalettePropagation()호출은 모든 자식창문부품들이 새로 변경한 QPalette를 사용할수 있도록 창문부품환경을 구성한다. 36행의 setPalette()호출은 새로운 조색판을 이 창문부품과 그의 모든 자식창문부품들에 의해서 사용되는것으로 확립한다.

38～47행은 그림 11-6에 보여준 창문부품배치를 만든다. 표식자는 그 본문에 표준전경색을 사용하므로 그것은 적색으로 된다. 누름단추의 본문은 백색으로 그려지고 단추의 배경색은 푸른색이다. 이 색들은 단추의 상태가 표준으로부터 변화될 때까지(마우스로 단추에 들어감으로써) 그 단추에서 유지된다.

그림 11－6. 오직 부모창문부품의 색만 변경

전달이 기정값 SamePalette로 지정하면 꼭같은 조색판을 가지는 모든 자식창문부품들의 조색판은 갱신된것으로 바뀐다. 이것은 일부 자식창문부품들에 새로운 조색판을 적용할수 있게 하지만 다른것들은 그대로 남겨둔다. 그러기 위하여 새로운 조색판을 만들고 그것을 수정할수 있게 하려는 모든 창문부품들과 부모창문부품에 할당한다. 그때부터 SamePalette설정을 사용하는 부모창문부품에 새로운 조색판을 할당하면 오직 그 지정된 창문부품들만 자기의 변경된 조색판들을 가지게 한다.

제7절. 사용자의 색만들기에 QPalette의 사용

어떤 형식의 저준위도형을 처리하는 자기의 창문부품을 만들 때는 색을 선택할 필요가 있다. 필요하면 사용하려는 정확한 색을 지정하거나 QPalette에 보관된것들을 꺼낼수 있다. 미리 정의된 QPalette색들을 사용하면 응용프로그람에서 색들이 변경될 때 자기 창문부품의 색들도 변화된다는 우점을 가진다.

다음 실례는 창문부품이 QPalette에 보관된 색을 사용하는 방법을 보여준다.

```
1 /* usepalette.cpp */
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include "usepalette.h"
5
6 int main(int argc, char **argv)
7 {
8     KApplication app(argc, argv,  "usepalette");
9     UsePalette usepalette;
10    usepalette.show();
11    app.setMainWidget(&usepalette);
12    return  (app.exec());
13 }
14 UsePalette::UsePalette(QWidget *parent,const
15    char *name) : QWidget(parent,name)
16 {
17    setFixedSize(375,250);
18 }
19 void UsePalette::paintEvent(QPaintEvent *)
20 {
21    QColorGroup group = colorGroup();
22    QColor midColor = group.color(QColorGroup::Mid);
23    QColor lightColor = group.color(QColorGroup::Light);
24    QBrush midBrush(midColor);
25    QBrush lightBrush(lightColor);
26    QPainter p;
27    p.begin(this);
28    p.fillRect(75,50,150,100,midBrush);
29    p.fillRect(150,100,150,100,lightBrush);
30    p.end();
31 }
```

UsePalette창문부품을 그려야 할 때마다 19행의 paintEvent호출이 있다.

21행의 QWidget메쏘드 colorGroup()호출은 현재의 QColorGroup을 되돌려준다. 되돌려진 실제색그룹은 창문부품의 상태에 따라 변화하므로 그룹으로부터 색을 사용하면 자기 창문부

품의 색들은 그 현재상태를 반영하여 변화된다.

22행과 23행은 QColorGroup의 color()을 호출하여 그리기에 사용할 2가지 색객체를 돌려준다. 그다음에 색들은 4행와 25행에서 사용되여 한쌍의 QBrush객체(매개 색에 하나씩)을 만든다. 이 객체들은 28행과 29행에서 그림 11-7에 보여준 직4각형들을 그리는데 사용된다. 이 paintEvent()메쏘드와 QPainter사용방법에 대한 자세한 정보는 12장에 있다.
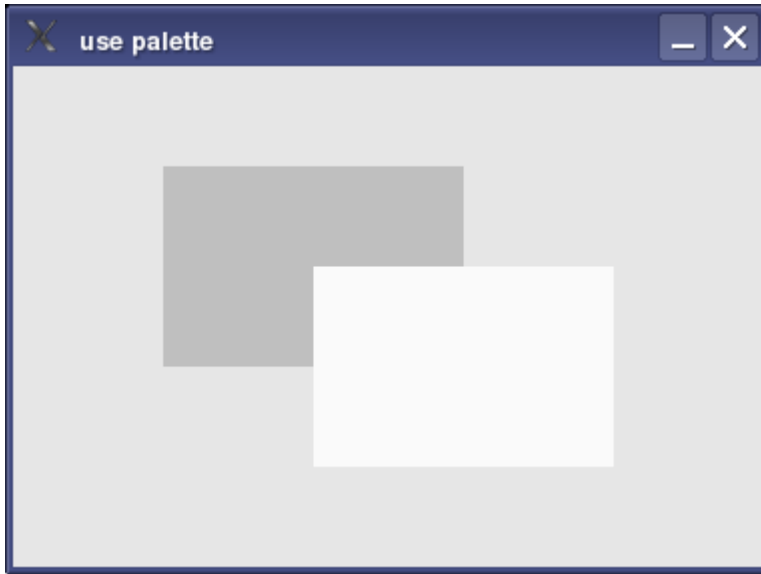
그림 11-7. 외부적으로 수정할수 있는 색들의 실례

요 약

KDE와 Qt에서 정의된 특별한 색부류의 우점들을 리용함으로써 완전한 응용프로그람의 색채가 풍부한 모양을 직접 조종, 설정, 그리고 변경할수 있다. 이 장은 다음과 같은것을 설명하였다.

- 사용할수 있는 색수는 주어진 체계의 하드웨어능력에 달려있다. 유한개의 사용가능색이 있을뿐아니라 임의의 시각에 보통 그 부분집합만 사용가능하다.
- QColor객체는 하드웨어에 의존하지 않는 색에 대한 원색정보를 가진다. QColor객체의 모양은 한 체계에서 다른 체계에로 넘어갈 때 변하지만 색들은 그 차이가 보통 문제로 되지 않을 정도로 아주 근사하다.
- 색들은 그룹으로 묶고 그룹들을 조색판에 수집한다. 그 모두는 응용프로그람의 매개 창문부품이 그 매개 부분을 그리거나 칠할 때 정확한 색을 쉽게 선택하도록 환경을 구성한다.
- 응용프로그람이 사용하는 색은 기정색모임으로부터 얻을수 있고 명백히 적-록-청수값들에 의해 정의되고 미리 정의된 색이름목록으로부터 얻거나 응용프로그람의 사용자가 환경의 부분으로 포함할수 있다.

제12장. QPainter에 의한 그리기와 색칠하기

학습내용
한번에 한 화소를 리용하여 도형화상을 그리기 직4각형과 타원과 같은 도형들을 그리고 색칠하기 단일직선과 복합직선의 그리기 호와 환4각형과 같은 특수도형만들기 현시창문에 픽스매프를 복사하기

이 장에서는 도형처리와 관련한 기술을 론의하며 화소, 직선, 곡선, 본문과 령역을 도색하는데 필요한 기본함수들의 실례들을 설명한다.

기본적인 도형처리기술은 물론 X11도형처리서고에 의존한다. Qt쏘프트웨어는 C++클라스집합안에 X11함수들을 포함하므로 X11에 익숙되면 모든것이 처음에는 좀 낡아보일수 있지만 QWidget에 직접 사용될수 있는 QPainter가 있다.

제1절. QPaintDevice에 화소를 그리기

QPaintDevice에 화소들을 그리거나 칠하는데 QPainter객체를 사용할수 있다. QPaintDevice는 표 12-1에 서술한 클라스들에 의해 계승되는 기초클라스이다.

QPainter객체는 그리기와 칠하기를 모두 수행한다. QPainter객체는 QPen객체와 QBrush객체를 포함한다. QPen객체는 화소그리기와 직선그리기

그리고 본문에, QBrush객체는 구역을 칠하는데 사용된다. 프로그람이 때때로 그것들을 바꿀수 있지만 오직 하나의 QPen과 하나의 QBrush가 임의의 시간에 QPainter객체내부에 존재하게 된다.

표 12-1. QPainter도형그리기지령을 받아들이는 클라스들

클라스	설명
QPicture	QPicture객체는 QPainter로부터 도형그리기지령을 받아들이고 그것들을 기록한다. 지령들은 그후에 다른 QPaintDevice객체에 그려질수 있으며 또한 후에 회복을 위해 파일에 써넣을수 있다.
QPixmap	QPainter는 QPixmap객체에 직접 그리거나 칠하는데 사용될수 있다. 현존 픽스매프도형을 변경하려고 하거나 그리려는 도형이 너무 복잡하지만 한번 수행하려고 한다면 이것을 리용한다.
QPrinter	QPrinter객체에 그리는 도형을 postscript로 변환되고 인쇄스풀러(lp, lpr, 또는 어떤것이든지)에 보내진다. QPrinter는 페지작성을 위한 메쏘드 즉 페지크기설정, 현재페지를 인쇄기에 보내기, 방향설정 등을 가지고있다.
QWidget	모든 현시가능객체는 QWidget이므로 얼마든지 현시가능객체에 직접 그리고 도색할수 있다. 빈 창문부품들은 창문부품이 이미 자기 소유(단추의 본문과

	같은)의 도형을 가지고있으면 창문부품의 도형은 자기의것과 충돌할수 있다.

이 장은 QPainter메쏘드들의 호출애 의한 도형처리를 설명하기 위하여 QWidget객체를 사용한다. 그리기 자체는 창문부품을 그려야(또는 다시 그려야) 할 때마다 호출되는 paintEvent()라는 메쏘드에서 수행된다. 이 메쏘드는 실행을 시작한 프로그람의 로출, 창문크기의 변경, 어두운 창문이 제거될 때마다 호출된다. 이 메쏘드를 호출할 때마다 창문은 이미 배경색으로 지워졌으므로 어떤 지우기도 필요없다. 프로그람이 하여야 할 일은 그림을 그리는것이다. 내부적으로 그리기는 이미 완충되여있고 화면이 갱신될 때 깜빡임이 없도록 현시기에 복사된다.

제2절. 직4각형

많은 도형들은 QPainter를 사용하여 그릴수 있으며 그것들을 그리는 방법은 수십가지 있다. 이 절은 간단한 직4각형을 그리고 도색하는 기본방법들의 실례를 제공함으로써 Qt도형처리의 기초를 보여준다. 다음 실례프로그람은 빈 창문부품을 만들고 그 창문에 직4각형을 그린다.

DrawRectangle머리부파일

```
1 /* drawrectangle.h */
2 #ifndef DRAWRECTANGLE_H
3 #define DRAWRECTANGLE_H
4
5 #include <qwidget.h>
6
7 class DrawRectangle: public QWidget
8 {
9 public:
10     DrawRectangle(QWidget *parent=0,const char *name=0);
11 protected:
12     virtual void paintEvent(QPaintEvent *);
13 };
14
15 #endif
```

DrawRectangle클라스는 응용프로그람의 제일 웃준위창문부품으로 사용되는 창문부품이다. 가상메쏘드 paintEvent()는 QWidget클라스의 메쏘드를 재정의하며 창문부품의 표면을 칠하거나 다시 칠할 필요가 있을 때마다 호출된다.

DrawRectangle

```
1 /* drawrectangle.cpp */
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include "drawrectangle.h"
5
6 int main(int argc, char **argv)
7 {
8      KApplication app(argc, argv,  "drawrectangle");
9      DrawRectangle drawrectangle;
10     drawrectangle.show();
11     app.setMainWidget(&drawrectangle);
12     return  (app.exec());
13 }
14 DrawRectangle::DrawRectangle(QWidget *parent,const
15     char *name) : QWidget(parent,name)
16 {
17     setFixedSize(400,200);
18 }
19 void DrawRectangle::paintEvent(QPaintEvent *)
20 {
21     QPainter p;
22     p.begin(this);
23     p.drawRect(50,50,300,100);
24     p.end();
25 }
```

14행에서 시작하는 구성자는 고정크기창문을 가지는 창문부품을 만든다. 도형은 오직 창문이 실현된 후에(즉 그것은 실제로 화소값들을 보관할 장소를 가진다) 현시기에 배치될 수 있기때문에 구성자에 의해 그려지는 그림은 없다.

paintEvent()메쏘드는 현시기를 재생하여야 할 때는 언제나 호출된다. 즉 창문은 사용자가 무엇을 그렸는가를 기억하지 않으며 자동적으로 그것을 교체한다. 프로그람은 창문을 다시 그릴 준비가 되여있어야 하며 이 메쏘드가 호출될 때마다 그것을 새로 그려야 한다. 보통 창문부품의 유일한 부분이 로출되여도 모든것을 그리는데 드는 실제비용은 없지만 도형이 복잡하고 도형을 그리는데 일정한 시간이 요구되면 그리기에 영향을 받는 구역만으로 제한할수 있다. 이러한 제한을 잘라내기(clipping)라고 부른다.

내부적으로 QPainter객체는 QPen객체를 사용하여 화소와 직선을 그린다. 기정QPen은 흑

색이고 1 화소너비로 선을 그린다.

19행의 메쏘드 paintEvent()은 그림 12-1에 보여주는 직4각형을 그린다. QPainter객체는 21행에서 만들어진다. 이 QPainter객체는 창문을 가지지 않으므로 그리는데 사용될수 없다. 22행의 begin()호출은 QPainter를 련결하고 직4각형의 그리기는 23행에서 drawRect()호출에 의해 발생한다. drawRect()에 넘긴 인수는 직4각형의 왼쪽웃구석의 x와 y자리표이고 그 뒤에 너비와 높이가 온다. 화상을 완전히 그렸을 때(이 경우에 간단한 직4각형) end()메쏘드호출이 이루어지고 QPainter와 QWidget가 분리된다.
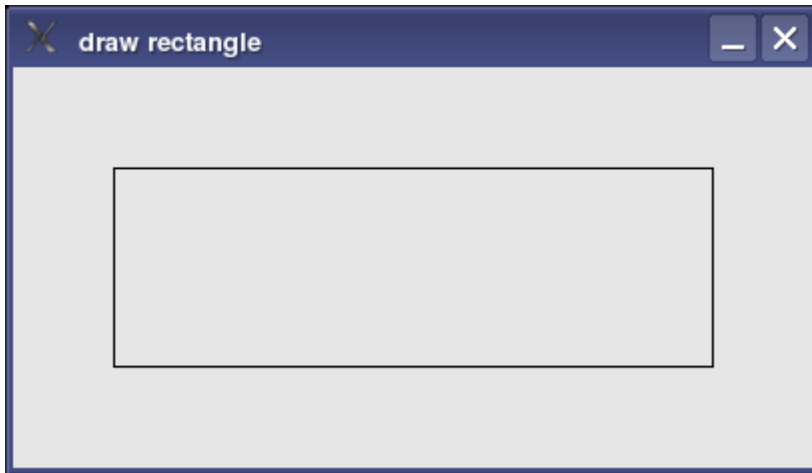
그림 12-1. 창문중심에 그려지는 직4각형

begin()과 end()메쏘드들을 모두 호출하여야 하고 구성자인수로서 그리기목표를 지정함으로써 구성자와 해체자가 그것을 하게 할수 있다. 다음 메쏘드는 이전것처럼 정확히 동작한다. 사용자가 어느것을 사용하는가는 개인적인 취미문제이다.

```
void DrawRectangle2::paintEvent(QPaintEvent *)
{
   QPainter p(this);
   p.drawRect(50,50,300,100);
}
```

이 실례의 구성자안에서 begin호출이 이루어지고 end()호출은 해체자안에서 이루어진다. (그것은 QPainter가 메쏘드의 끝에서 범위밖으로 벗어날 때 자동적으로 호출된다.)

알아두기: begin()메쏘드가 호출될 때마다 QPainter객체는 완전히 초기화되므로 펜과 솔들로 QPainter를 미리 설치한 다음 그것을 리용하여 여러 창문부품들을 그리는 방법은 없다.

직4각형의 칠하기는 그것을 그리는것과 좀 다르다. 다음 실례는 이전 실례와 같은 기본치수들을 사용하지만 그림 12-2에 보여준 색칠한 직4각형을 만든다.

```
1 /* fillrectangle.cpp */
2 #include <kapplication.h>
```

```
3 #include <qpainter.h>
4 #include "fillrectangle.h"
5
6 int main(int argc, char **argv)
7 {
8     KApplication app(argc, argv,  "fillrectangle");
9     DrawRectangle fillrectangle;
10    fillrectangle.show();
11    app.setMainWidget(&fillrectangle);
12    return  (app.exec());
13 }
14 DrawRectangle::DrawRectangle(QWidget *parent,const
15    char *name) : QWidget(parent,name)
16 {
17    setFixedSize(400,200);
18 }
19 void DrawRectangle::paintEvent(QPaintEvent *)
20 {
21    QBrush brush(QColor( “black” ));
22    QPainter p;
23    p.begin(this);
24    p.fillRect(50,50,300,100,brush);
25    p.end();
26 }
```
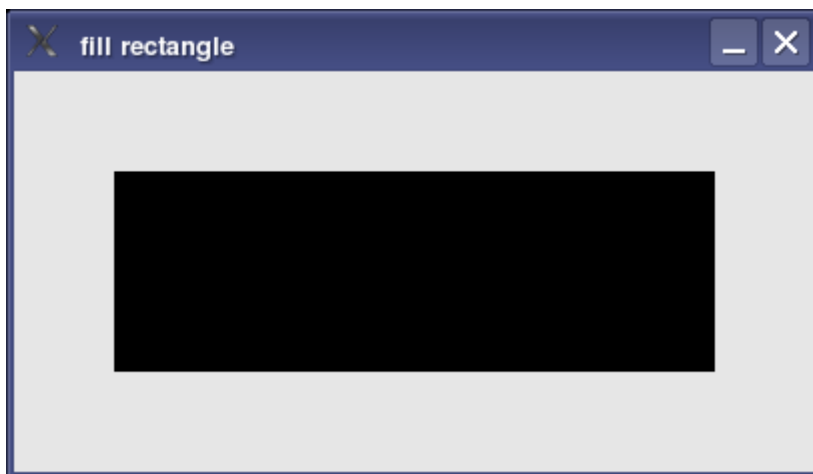
그림 12-2. 창문중심의 색칠한 직4각형

그리기는 19행에서 시작하는 paintEvent()에서 실행된다. 그리기는 QPen객체(기정은 흑색펜)을 가지고 수행되지만 도색은 솔(기정솔은 칠하지 않는다)을 가지고 수행한다. 그러므로 paintEvent()메쏘드의 21행은 흑색화소들로 구역의 모두를 도색하는데 사용할 QBrush객체를 만든다.

QPainter객체는 22행에서 만들어지고 23행에서 창문부품에 련결된다. 24행의 fillRect()호출은 초기의 drawRect()와 같이 직4각형의 왼쪽웃구석의 x와 y자리표와 직4각형의 너비와 높이를 요구한다. 그것은 또한 직4각형을 칠하는데 사용하는 솔을 요구한다.

직4각형의 그리기와 도색을 모두 할수 있다. 다음 실례는 우선 흰솔로 직4각형을 도색한 다음 기정흑색펜으로 백색구역의 륜곽을 그린다.

```
void FillRectangle2::paintEvent(QPaintEvent *)
{
    QBrush brush(QColor("white"));
    QPainter p;
    p.begin(this);
    p.fillRect(50,50,300,100,brush);
    p.drawRect(50,50,300,100);
    p.end();
}
```

결과는 그림 12-3에 보여준다. 직4각형은 그리는 직4각형의 외부경계가 색칠하는 직4각형의 외부경계와 정확히 일치하므로 그리기전에 도색되여야 하며 직4각형을 도색하는 동작은 이미 거기에 있는것은 모두 지워버린다.
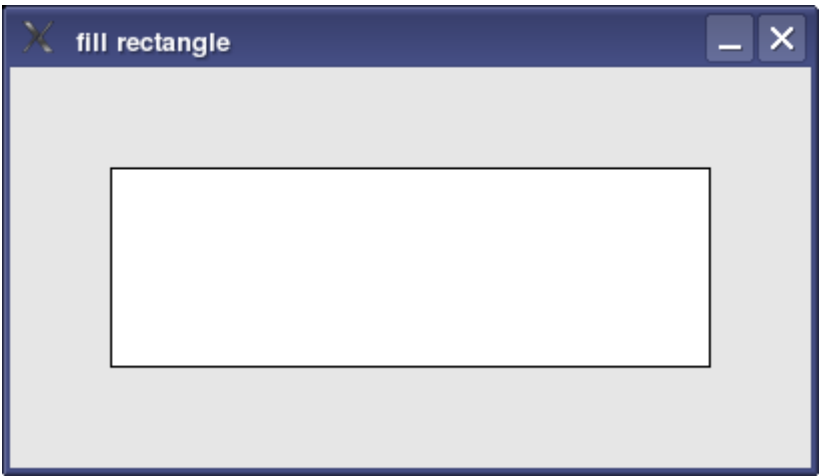
그림 12-3. 직4각형의 도색과 륜곽그리기

다른 수법도 있다. 모든 그리기메쏘드는 도형륜곽그리기와 칠하기에 모두 사용될수 있다. 륜곽그리기에는 펜이 사용되지만 칠하기에는 솔이 사용된다. 그림 12-3에 보여준 직4각형은 또한 다음 paintEvent()메쏘드에 의해 생성될수 있다.

```
void FillRectangle3::paintEvent(QPaintEvent *)
{
   QBrush brush(QColor("white"));
   QPainter p;
   p.begin(this);
   p.setBrush(brush);
   p.drawRect(50,50,300,100);
   p.end();
}
```

이 실례에서 기정QPen(흑색, 가는 선)은 륜곽선을 그리는데 사용되지만 기정QBrush는 없으므로 그것을 만들어서 도형을 칠하기전에 QPainter에 할당하여야 한다. drawRect()호출은 우선 솔색으로 직4각형구역을 칠하고 그다음에 펜으로 그 륜곽을 그린다.

제3절. 펜

QPen으로 선을 그리기 위하여서는 3가지 속성 즉 색, 너비, 그리고 점 또는 사슬패턴을 가지고 있어야 한다. 기정펜은 가장 작은 선(1화소너비의 선)을 그리는 0의 너비를 가지고있다. 이것은 대체로 선너비를 1로 설정한것과 같다. 0의 화소너비로 그린 선은 그림의 확장에 관계없이 정확히 1화소너비를 유지한다. 그렇지만 1이상의 화소너비로 그린 선은 그림을 확대 혹은 축소할 때 그 두께가 변한다.

다음 실례는 그림 12-4에 보여준 창문을 현시한다. 그것은 흑색으로 2화소너비의 6가지 사용가능한 형식의 직선을 그린다. 첫째 형식(NoPen)은 그 어떤 선도 그리지 않는다. 실례로 직4각형내부를 칠하고 직4각형의 륜곽을 그리지 않으려면 NoPen형식을 사용할수 있다.
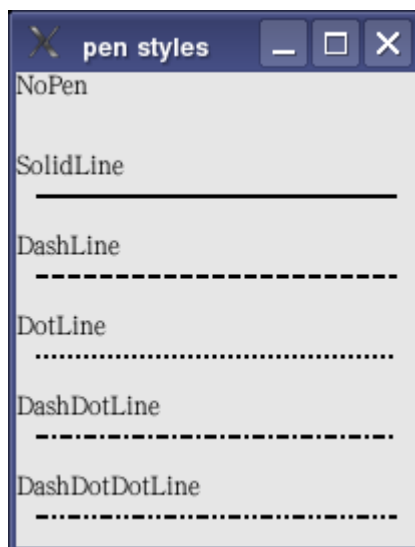
그림 12-4. 6가지 QPen직선형식

알아두기: 이 실례는 drawLine()을 사용하여 각이한 직선형식을 보여주지만 직선형식은 drawArc(), drawRect() 그리고 drawPolyline()를 비롯하여 모든 그리기메쏘드에 적용된다.

PenStyles머리부파일

```
1 /* penstyles.h */
2 #ifndef PENSTYLES_H
3 #define PENSTYLES_H
4
5 #include <qwidget.h>
6 #include <qlabel.h>
7
8 class PenStyles: public QWidget
9 {
10 public:
11     PenStyles(QWidget *parent=0,const char *name=0);
12 private:
13     QLabel *label[6];
14     QWidget *widget[6];
15 protected:
16     virtual void paintEvent(QPaintEvent *);
17 };
18
19 #endif
```

PenStyles클라스는 현시에 사용되는 표식자와 창문부품들의 배렬들을 포함한다. paintEvent()역호출메쏘드가 창문부품들의 호출을 요구하기때문에 창문부품들의 배렬을 클라스의 부분으로서 포함할 필요가 있다.

PenStyles

```
1 /* penstyles.cpp */
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include <qlayout.h>
5 #include "penstyles.h"
6
7 struct pstyleStruct {
8     QString name;
9     Qt::PenStyle style;
```

```
10 } pstyle[6] = {
11    { "NoPen", Qt::NoPen },
12    { "SolidLine", Qt::SolidLine },
13    { "DashLine", Qt::DashLine },
14    { "DotLine", Qt::DotLine },
15    { "DashDotLine", Qt::DashDotLine },
16    { "DashDotDotLine", Qt::DashDotDotLine }
17 };
18
19 int main(int argc, char **argv)
20 {
21    KApplication app(argc, argv,” penstyles” );
22    PenStyles penstyles;
23    penstyles.show();
24    app.setMainWidget(&penstyles);
25    return  (app.exec());
26 }
27 PenStyles::PenStyles(QWidget *parent,const
28    char *name) : QWidget(parent,name)
29 {
30    QVBoxLayout *vbox = new QVBoxLayout(this,0,3);
31
32    for(int i=0; i<6; i++) {
33            label[i] = new QLabel(pstyle[i].name,this);
34            vbox->addWidget(label[i]);
35            widget[i] = new QWidget(this);
36            widget[i]->setFixedHeight(20);
37            widget[i]->setFixedWidth(200);
38            vbox->addWidget(widget[i]);
39    }
40    resize(10,10);
41 }
42 void PenStyles::paintEvent(QPaintEvent *)
43 {
44    QColor black("black");
```

```
45    QPainter p;
46    for(int i=0; i<6; i++) {
47            p.begin(widget[i]);
48            QPen pen(black,2,pstyle[i].style);
49            p.setPen(pen);
50            p.drawLine(10,5,190,5);
51            p.end();
52    }
53 }
```

7∼17행의 구조체배렬은 편리상 간단히 각이한 직선형식들의 이름과 값들을 보관한다. 형식들의 이름은 qnamespace.h에서 정의되는데 그것은 Qt클라스를 선언한다. QObject와 일부 다른 중요한 클라스들이 Qt를 계승하므로 공개적으로 그것을 포함할 필요는 거의 없다.

27행에서 시작하는 구성자는 수직칸을 리용하여 12개 창문부품의 렬을 가진다. 6개 QLabel창문부품과 6개 일반적인 QWidget창문부품이 있다. 표식자들은 직선형식의 이름을 현시하는데, 그리고고 QWidget는 선을 현시하는데 사용된다. 32행에서 시작하는 순환은 표식자와 창문부품들을 모두 만들고 그 크기의 설정, 수직칸에 그것들의 추가, 클라스의 배렬에 그것들의 참고를 보관한다.

직선의 실제그리기는 창문을 그려야 할 때마다 호출되는 24행의 paintEvent()메쏘드에서 수행된다. 나머지 일은 직선을 그리는것이다. 46행에서 시작하는 순환은 6개 창문부품의 창문에 단일직선을 그린다. 그리기를 위하여 47행에서 begin()호출이 이루어지고 직선을 받아들이는 창문부품에 QPainter객체를 할당한다. 48행은 적합한 형식의 QPen을 만들고 49행은 setPen()을 호출하여 모든 그리기에서 사용할 새로운 펜을 설정한다. 직선은 50행의 drawLine()호출에 의해 그려진다. 순환의 끝에서 end()메쏘드가 호출되여 이 창문부품과 QPainter를 분리하므로 그것은 순환고리의 꼭대기에서 배렬안의 다음 창문부품에 사용될수 있다.

> 알아두기: QLabel창문부품의 본문은 구성자에서 삽입되지만 메쏘드 paintEvent()를 호출할 때까지 아무것도 그려지지 않는다. 사실상 프로그람이 표식자에 본문을 삽입하더라도 그것은 실제로 paintEvent()에로의 역호출이 있을 때까지 그리지 않는다.

제4절. 표준솔

QBrush로 령역을 칠하기 위하여서는 2가지 속성 즉 색과 패턴이 있어야 한다. 기정색은 흑색이고 기정패턴은 SolidPattern이다. 15가지의 미리 정의된 패턴을 사용할수 있지만 또한 자체로 만들수 있다. 다음 프로그람은 그림 12-5에 보여준것과 같이 미리 정의된 패턴들을 현시한다.
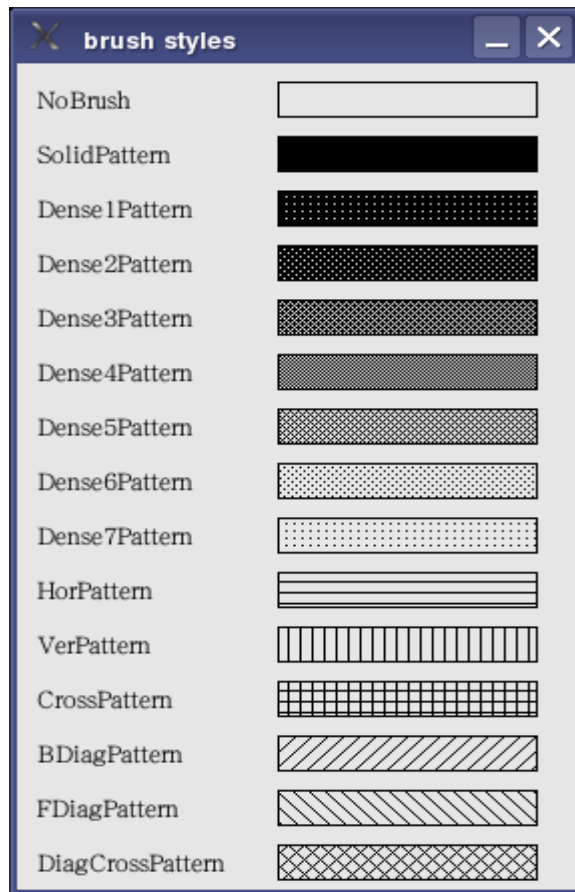
그림 12-5. 15개의 미리 정의한 솔형식

BrushStyles머리부파일

```
1 /* brushstyles.h */
2 #ifndef BRUSHSTYLES_H
3 #define BRUSHSTYLES_H
4
5 #include <qwidget.h>
6
7 class BrushStyles: public QWidget
8 {
9 public:
10     BrushStyles(QWidget *parent=0,const char *name=0);
11 protected:
12     virtual void paintEvent(QPaintEvent *);
13 };
14
```

```
15 #endif
```

BrushStyles클라스는 빈 창문부품이므로 그 창문에 본문과 색칠한 직4각형들을 둘다 그릴수 있다.

BrushStyles

```
1 /* brushstyles.cpp */
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include <qlayout.h>
5 #include "brushstyles.h"
6
7 struct bstyleStruct {
8     QString name;
9     Qt::BrushStyle style;
10 } bstyle[15] = {
11    { "NoBrush", Qt::NoBrush },
12    { "SolidPattern", Qt::SolidPattern },
13    { "Dense1Pattern", Qt::Dense1Pattern },
14    { "Dense2Pattern", Qt::Dense2Pattern },
15    { "Dense3Pattern", Qt::Dense3Pattern },
16    { "Dense4Pattern", Qt::Dense4Pattern },
17    { "Dense5Pattern", Qt::Dense5Pattern },
18    { "Dense6Pattern", Qt::Dense6Pattern },
19    { "Dense7Pattern", Qt::Dense7Pattern },
20    { "HorPattern", Qt::HorPattern },
21    { "VerPattern", Qt::VerPattern },
22    { "CrossPattern", Qt::CrossPattern },
23    { "BDiagPattern", Qt::BDiagPattern },
24    { "FDiagPattern", Qt::FDiagPattern },
25    { "DiagCrossPattern", Qt::DiagCrossPattern }
26 };
27
28 int main(int argc, char **argv)
29 {
30    KApplication app(argc, argv,  "brushstyles");
31    BrushStyles brushstyles;
```

```
32    brushstyles.show();
33    app.setMainWidget(&brushstyles);
34    return  (app.exec());
35 }
36 BrushStyles::BrushStyles(QWidget *parent,const
37    char *name) : QWidget(parent,name)
38 {
39    setFixedSize(280,455);
40 }
41 void BrushStyles::paintEvent(QPaintEvent *)
42 {
43    int xText = 10;
44    int xFill = 130;
45    int yText = 25;
46    int yFill = 10;
47    QColor black("black");
48    QPainter p(this);
49    for(int i=0; i<15; i++) {
50            QBrush brush(black,bstyle[i].style);
51            p.setBrush(brush);
52            p.drawText(xText,yText,bstyle[i].name);
53            p.drawRect(xFill,yFill,130,20);
54            yText += 30;
55            yFill += 30;
56    }
57 }
```

7～26행에서 정의된 구조체배렬은 미리 정의된 15가지 형식들의 개개의 이름과 값들을 보관하기 위한것이다. 형식이름은 qnamespace.h에서 정의된다.

36행에서 시작하는 구성자는 본문과 직4각형을 모두 포함하는 크기가 고정인 창문을 설정한다.

41～57행의 paintEvent()메쏘드는 창문을 색칠할 필요가 있을 때마다 호출된다. xText와 yText값들은 본문의 첫 글자위치를 지정하지만 xFill과 yFill은 색칠하려는 직4각형의 왼쪽웃구석의 위치를 결정한다. 49행에서 시작하는 순환은 매개 색칠형식에 대하여 한번씩 실행한다. 50행은 미리 정의된 형식들중에서 하나를 선택하여 QBrush객체를 만들고 51행은 QPainter에 그것을 설정한다. 52행에서 drawText()호출에 의해 본문을 그린다. 직4각형구역은

53행에서 drawRect()호출에 의해 색칠해진다. 직4각형은 기정흑색QPen이 여전히 유효하므로 흑색으로 색칠되고 륜곽이 그려지며 새 견본의 QBrush가 추가된다.

제5절. 사용자정의솔만들기

령역을 채우기 위하여 QBrush는 작은 픽스매프를 만들고 창문에 그것을 타일모양으로 붙인다. 그러므로 자체의 색칠패턴을 QPixmap형식으로 지정하는것은 간단한 문제이다. 다음 프로그람은 창문의 직4각형령역을 칠하기 위하여 픽스매프를 사용한다.

BrushCustom

```
1 /* brushcustom.cpp */
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include <qlayout.h>
5 #include "brushcustom.h"
6
7 static const char *mypattern[] = {
8     "16 16 4 1",
9     " c blue",
10    ". c white",
11    "x c red",
12    "y c green",
13    "yy.... ....yy",
14    "yy.... ....yy",
15    "...... ......",
16    "...... ......",
17    "...... ......",
18    "...... ......",
19    " ..... x       ",
20    " .... xxx      ",
21    " ... xxxxx    ",
22    " .. xxxxxxx   ",
23    ". ..xxxxxxxxx",
24    ".. ..........",
25    "... .........",
26    ".... ........",
27    "yy... .....yy",
```

```
28   "yy.... ....yy"
29 };
30
31 int main(int argc, char **argv)
32 {
33    KApplication app(argc, argv,"brushcustom");
34    BrushCustom brushcustom;
35    brushcustom.show();
36    app.setMainWidget(&brushcustom);
37    return  (app.exec());
38 }
39 BrushCustom::BrushCustom(QWidget *parent,const
40    char *name) : QWidget(parent,name)
41 {
42    setFixedSize(220,120);
43 }
44 void BrushCustom::paintEvent(QPaintEvent *)
45 {
46    QBrush brush;
47    QPixmap pixmap(mypattern);
48    brush.setPixmap(pixmap);
49
50    QPainter p(this);
51    p.setBrush(brush);
52    p.drawRect(20,20,180,80);
53 }
```

려과기로 사용될 픽스매프는 7～29행에 XPM형식으로 있다.

참고: 9장은 XPM형식과 그 사용방법에 대하여 설명하였다.

39행에서 시작하는 구성자는 120×220화소의 고정크기로 창문부품의 창문을 설정한다.

그리기는 44행의 paintEvent()에서 진행된다. 47행은 픽스매프정보를 사용하여 QPixmap객체를 만들고 48행은 setPixmap()를 호출하여 QBrush에 픽스매프를 설정한다. 픽스매프설치에서는 솔형식을 CustomPattern으로 설정하는데 이 값은 오직 픽스매프에만 사용되므로 앞의 실례에 포함되지 않았다. 이 실례가 생성하는 창문은 그림 12-6에 보여주었다. 일단 솔이 정의되고 QPainter에 설정되면 그것을 다른것으로 변경할 때까지 색칠에 사용한다.
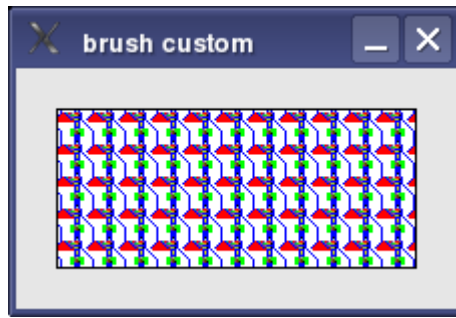
그림 12-6. 사용자정의솔을 리용한 직4각형칠하기

타일붙이기는 그리고있는 실제도형의 위치에 관계없이 수행된다. 이 장의 마지막에 설명하겠지만 원점은 이동될수 있으나 왼쪽웃구석우의 기정원점은 첫째 타일이 놓이는곳이고 다른 모든 타일은 그 린접에 붙여진다. 원점타일을 볼수 있게 할 필요는 없지만 모든 다른 타일들의 위치는 그로부터 평가된다. 그림 12-6을 보고 프로그람의 XPM자료와 그것을 비교한다면 왼쪽웃타일의 왼쪽웃부분이 잘리운것을 볼수 있다.

제6절. QPaintDevice의 측도

창문크기가 변할 때 창문에 그려진 도형의 크기가 변경되거나 환경에 적응되여야 할 필요가 자주 생긴다. 클라스 QPaintDeviceMetrics는 QPaintDevice의 현재크기(및 기타 정보)를 결정하는데 사용한다. 다음 실례프로그람은 QPaintDeviceMetrics객체의 정보를 사용하여 창문에 도색하는 도형들의 위치와 크기를 변화시킨다. 그림 12-7과 12-8에 보여준것처럼 본문은 창문에서 수직방향으로 균일한 간격을 유지하며 배경에서 흰직4각형은 창문에 맞게 그자체의 모양을 변경한다.
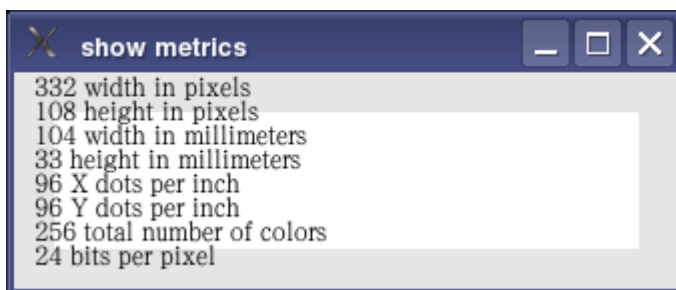
그림 12-7. 본문이 창문에 적합하게 가까이 이동한 실례
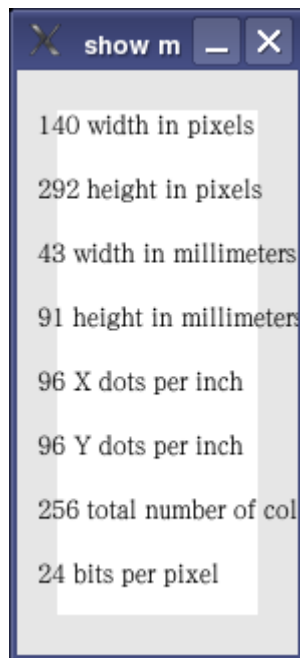
그림 12-8. 본문이 창문을 채우기 위하여 더 멀리 이동한 실례

ShowMetrics머리부파일

```
1 /* showmetrics.h */
2 #ifndef SHOWMETRICS_H
3 #define SHOWMETRICS_H
4
5 #include <qwidget.h>
6
7 class ShowMetrics: public QWidget
8 {
9 public:
10     ShowMetrics(QWidget *parent=0,const char *name=0);
11 protected:
12     virtual void paintEvent(QPaintEvent *);
13 };
14
15 #endif
```

ShowMetrics

```
1 /* showmetrics.cpp */
2 #include <kapplication.h>
```

```
3 #include <qpainter.h>
4 #include <qpaintdevicemetrics.h>
5 #include "showmetrics.h"
6
7 int main(int argc, char **argv)
8 {
9     KApplication app(argc, argv,  "showmetrics");
10    ShowMetrics showmetrics;
11    showmetrics.show();
12    app.setMainWidget(&showmetrics);
13    return  (app.exec());
14 }
15 ShowMetrics::ShowMetrics(QWidget *parent,const
16    char *name) : QWidget(parent,name)
17 {
18    resize(400,300);
19 }
20 void ShowMetrics::paintEvent(QPaintEvent *)
21 {
22    QString str;
23    QPaintDeviceMetrics metrics(this);
24    QBrush brush(QColor("white"));
25    int yincr = metrics.height() / 9;
26    int y = yincr;
27    int x = 10;
28
29    QPainter p(this);
30    if((metrics.width() > 40) && (metrics.height() > 40)) {
31            p.setPen(Qt::NoPen);
32            p.setBrush(brush);
33            p.drawRect(20,20,
34            metrics.width() - 40,metrics.height() - 40);
35            p.setPen(Qt::SolidLine);
36            p.setBrush(Qt::NoBrush);
37    }
```

```
38    str.sprintf("%d width in pixels",
39    metrics.width());
40    p.drawText(x,y,str);
41    y += yincr;
42    str.sprintf("%d height in pixels",
43          metrics.height());
44    p.drawText(x,y,str);
45    y += yincr;
46    str.sprintf("%d width in millimeters",
47          metrics.widthMM());
48    p.drawText(x,y,str);
49    y += yincr;
50    str.sprintf("%d height in millimeters",
51          metrics.heightMM());
52    p.drawText(x,y,str);
53    y += yincr;
54    str.sprintf("%d X dots per inch",
55          metrics.logicalDpiX());
56    p.drawText(x,y,str);
57    y += yincr;
58    str.sprintf("%d Y dots per inch",
59          metrics.logicalDpiY());
60    p.drawText(x,y,str);
61    y += yincr;
62    str.sprintf("%d total number of colors",
63          metrics.numColors());
64    p.drawText(x,y,str);
65    y += yincr;
66    str.sprintf("%d bits per pixel",
67          metrics.depth());
68    p.drawText(x,y,str);
69 }
```

20행의 paintEvent()메쏘드는 창문을 칠할 필요가 있을 때마다 호출된다. 이것은 또한 창문이 처음으로 나타날 때, 창문이 그 앞에 있던 창문을 제거하여 로출될 때, 이 응용프로그람에서 가장 중요한 창문크기가 변화할 때마다 발생한다.

QPaintDeviceMetrics객체는 23행에서 만들어지는데 그것은 도형을 비례에 따라 확대축소하는데 필요한 모든 정보를 가진다. 여기에 2개의 높이와 너비값들이 있다. 하나는 화소수로 측정되고 다른것은 mm로 측정된다. 단위설정은 칠해지는 장치의 특성에 따른다. 실례로 인쇄기에 그림을 그리려한다면 mm를 단위로 리용하는것이 좋다. 그러나 현시된 창문에 만들어진 그림들은 크기(심지어 같은 화소분해능)에 따라 널리 변화할수 있으므로 화소측도를 가지고 조작하는것이 제일 좋다.

25～27행은 현시하려는 첫 문자렬의 x와 y자리표들을 설정하고 창문 아래로 문자렬들사이의 간격을 유지하기 위하여 y값을 변경할 증분을 계산한다. 8개 문자렬을 현시하려고 하므로 우와 아래를 계산하면 9개의 수직간격이 있으며 창문의 전체 높이를 9로 나누어 그들사이의 거리를 결정한다.

30～37행의 코드는 창문에 맞게 크기조절된 백색 직4각형을 그린다. 직4각형은 창문의 4개의 모든 변들로부터 정확히 20화소 떨어지도록 그려진다. 30행의 if문은 직4각형을 그릴 충분한 자리가 없으면 직4각형을 뛰여넘는다. 직4각형륜곽을 제한하기 위하여 31행의 setPen()호출은 현재 펜을 삭제한다. 32행의 setBrush()호출은 직4각형을 채울 흰솔을 설치하고 33행의 drawRect()호출은 직4각형을 그린다. 직4각형은 창문에 매 변으로부터 20화소 떨어진 곳에 만들어진다. 35행과 36행은 기정펜을 되살리고 흰솔을 삭제한다.

38～68행은 창문에서 수값과 본문을 인쇄한다. x값을 변화시키지 않고 모든 문자렬이 창문의 왼쪽변으로부터 같은 거리만큼 떨어져서 시작된다. 매개 선이 그려진 후에 yincr에 보관된 값은 매개 문자렬이 전체 창문높이의 9분의 1로서 우의 문자렬아래에 나타나도록 하기 위하여 y에 더해진다.

제7절. 화소그리기

다음 프로그람은 살창을 만들고 그우에 씨누스곡선을 그려서 그림 12-9와 같이 한번에 한 화소 그리기를 진행한다.
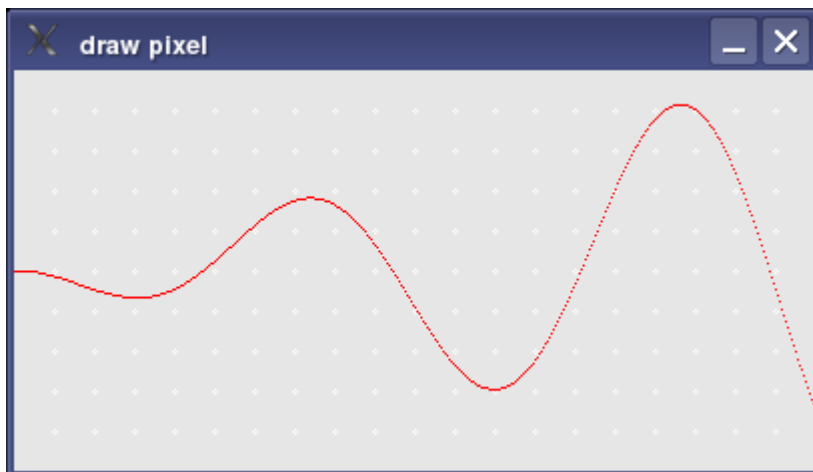
그림 12-9. 한번에 한화소씩 그린 곡선과 살창

DrawPixel머리부파일

```
1 /* drawpixel.h */
2 #ifndef DRAWPIXEL_H
3 #define DRAWPIXEL_H
4
5 #include <qwidget.h>
6
7 class DrawPixel: public QWidget
8 {
9 public:
10     DrawPixel(QWidget *parent=0,const char *name=0);
11 protected:
12     virtual void paintEvent(QPaintEvent *);
13 };
14
15 #endif
```

DrawPixel

```
1 /* drawpixel.cpp */
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include "drawpixel.h"
5
6 int main(int argc, char **argv)
7 {
8     KApplication app(argc, argv,  "drawpixel");
9     DrawPixel drawpixel;
10     drawpixel.show();
11     app.setMainWidget(&drawpixel);
12     return(app.exec());
13 }
14 DrawPixel::DrawPixel(QWidget *parent,const
15     char *name) : QWidget(parent,name)
16 {
17     setFixedSize(400,200);
18 }
```

```
19 void DrawPixel::paintEvent(QPaintEvent *)
20 {
21     QPainter p(this);
22     p.setPen(QColor("white"));
23     for(int x=20; x<400; x += 20) {
24         for(int y=20; y<200; y += 20) {
25             p.drawPoint(x-1,y);
26             p.drawPoint(x+1,y);
27             p.drawPoint(x,y-1);
28             p.drawPoint(x,y+1);
29         }
30     }
31     p.setPen(QColor("red"));
32     for(double x=0; x<400; x++) {
33         double y = sin(x / 30);
34         y *= x / 4;
35         y += 100;
36         p.drawPoint((int)x,(int)y);
37     }
38 }
```

19행에서 시작하는 paintEvent()메쏘드는 살창점들과 곡선을 그린다. 여기서 점은 보통 선을 그리는데 사용한 QPen으로 그린다. 화소는 가능한 모든 선들중에서 가장 짧은 선으로서 볼수 있다. 22행은 setPen()을 호출하여 점들을 그리기 위한 백색펜을 설정하고 31행은 setPen()을 호출하여 곡선을 만드는 점들을 그리기 위한 적색펜을 만든다.

23～30행의 순환은 그림 12-9에 보여준 백색점들을 그린다. 점들은 수직뿐아니라 수평으로도 모두 20화소간격으로 그려진다. 매개 점은 4개 화소로서 그려진다.

32～37행에서 순환은 왼쪽에서 오른쪽으로 크기가 증가하는 씨누스파를 그린다. 변수 x와 y는 계산을 간단화하기 위하여 2배로 선언한다. 창문은 400화소너비로 고정되므로 x값은 0에서 400까지 변화하고 400개의 《화소렬》에 1개의 색칠한 화소가 생긴다. 33행은 씨누스값을 계산한다. 그리고 x값을 라디안수로서 취급한다. (30이 아닌 나누는수를 사용하는것은 여기서 창문에 나타나는 주기수를 변경한다). 34행에 의해 x가 커질 때 y가 커진다. 35행은 y값에 100을 더하여 창문에서 수직으로 중심에 배치된다. 36행의 drawPoint()호출은 화소를 칠한다.

제8절. 화소배렬의 그리기

앞의 실례에서는 창문을 칠할 때마다 매번 모든 점들을 계산하였다. 보통 한번만 점들을 계산하거나 혹은 파일로부터 그것들을 적재하여 배렬에 보관하는것이 더 편리하다. 다음 실례는 그림 12-9와 같은 창문을 현시하지만 한번만 화소위치들을 계산하여 배렬에 보관한다.

DrawPixel2머리부파일

```
1 /* drawpixel2.h */
2 #ifndef DRAWPIXEL_H
3 #define DRAWPIXEL_H
4
5 #include <qwidget.h>
6 #include <qpointarray.h>
7
8 class DrawPixel2: public QWidget
9 {
10 public:
11     DrawPixel2(QWidget *parent=0,const char *name=0);
12 private:
13     QPointArray *grid;
14     QPointArray *curve;
15 protected:
16     virtual void paintEvent(QPaintEvent *);
17 };
18
19 #endif
```

13행과 14행은 한쌍의 QPointArray객체에로의 지적자들을 선언한다. curves는 궤도를 점들로 그리는데 사용되며 grid는 배경에서 흰점의 위치들에 살창을 그리는데 리용된다.

DrawPixel2

```
1 /* drawpixel2.cpp */
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include "drawpixel2.h"
5
6 int main(int argc, char **argv)
```

```
7 {
8     KApplication app(argc, argv,  "drawpixel2");
9     DrawPixel2 drawpixel2;
10    drawpixel2.show();
11    app.setMainWidget(&drawpixel2);
12    return  (app.exec());
13 }
14 DrawPixel2::DrawPixel2(QWidget *parent,const
15    char *name) : QWidget(parent,name)
16 {
17    int index;
18    setFixedSize(400,200);
19
20    grid = new QPointArray(4 * 20 * 10);
21    index = 0;
22    for(int x=20; x<400; x += 20) {
23            for(int y=20; y<200; y += 20) {
24                    grid->setPoint(index++,x-1,y);
25                    grid->setPoint(index++,x+1,y);
26                    grid->setPoint(index++,x,y-1);
27                    grid->setPoint(index++,x,y+1);
28            }
29    }
30    curve = new QPointArray(400);
31    index = 0;
32    for(double x=0; x<400; x++) {
33            double y = sin(x / 30);
34            y *= x / 4;
35            y += 100;
36            curve->setPoint(index++,(int)x,(int)y);
37    }
38 }
39 void DrawPixel2::paintEvent(QPaintEvent *)
40 {
41    QPainter p(this);
```

```
42    p.setPen(QColor("white"));
43    p.drawPoints(*grid);
44    p.setPen(QColor("red"));
45    p.drawPoints(*curve);
46 }
```

14행에서 시작하는 구성자는 계산을 수행하고 배렬들에 그 결과를 보관한다. 18행의 setFixedSize()호출은 창문이 크기가 변경되는것을 금지한다.

살창점을 가지기 위한 QPointArray객체는 20행에서 만들어진다. 배렬에는 매개 점에 대하여 한개 항목이 있으므로 배렬의 총 크기는 4(매개 살창점에서 화소수), 20(x축을 따라 나타나는 살창점의 수), 그리고 10(y축을 따라 나타나는 살창점의 수)의 적이다. 22～29행의 순환은 매개 살창점들에 4화소위치를 삽입한다.

곡선궤도를 포함하기 위한 QPointArray객체는 30행에서 만들어진다. 계산과 점의 개수는 앞의 실례와 같다. 400개의 점을 계산하였으므로 이것들은 36행의 setPoint()호출에 의해 배렬에 보관된다.

39행에서 시작되는 paintEvent()메쏘드는 앞의 실례보다 훨씬 적게 실행한다. QPainter객체가 만들어지고 백색펜은 살창의 점들을 그리는데 사용되며 적색펜은 곡선의 점들을 그리는데 사용된다.

일부 경우에 값들을 다시 계산할 필요가 있으나 그럴 필요가 없는 경우도 있다. 실례로 창문크기가 변할 때 값들을 다시 계산하려고 한다면 paintEvent()메쏘드의 제일 웃부분에서 QPaintDeviceMetrics의 값들을 사용하여 창문크기가 변화하였는가를 결정하고 만일 그렇다면 계산을 수행하는 메쏘드를 호출한다.

제9절. 벡토르직선그리기

2개의 메쏘드를 벡토르그리기에 사용할수 있다. 그러나 drawLine()으로 모든 그리기를 할수 있으므로 일정한 종류의 그림을 그리는데 매우 편리하다. 메쏘드 moveTo()와 lineTo()는 도형처리가 펜작도기에서 리용되던 방법이 그대로 적용된 메쏘드이다. 이 두 메쏘드는 한곳에서 다른 곳으로 펜을 이동하시켜 직선을 그리지만 그것들중 하나는 펜을 눌러서 직선을 그린다. 펜은 항상 위치를 가지므로 직선을 그리기 위하여서는 직선의 다른 끝을 지정하는 것만이 필요하며 일단 직선을 그렸으면 펜은 다시 새로운 위치를 지정하여야 한다.

다음 실례는 파일로부터 그리기지령을 읽어들이고 그것을 리용하여 그림 12-10에 보여준 도형을 현시한다. 입력본문파일의 매개 직선은 연산코드(이동은 m, 그리기는 d)와 사건이 발생하는 자리표점위치를 가진다. 이 실례에서 사용된 파일은 다음과 같은 파라메터를 가지고 시작된다.

```
m 60 110
d 60 10
```

d 160 10

d 160 60

m 160 80

d 160 180

...

첫째 명령은 점(60,110)으로 이동하는 지령이다. 둘째 명령은 펜의 위치 (60,110)로부터 새 위치 (60,10)까지 직선을 그린다.
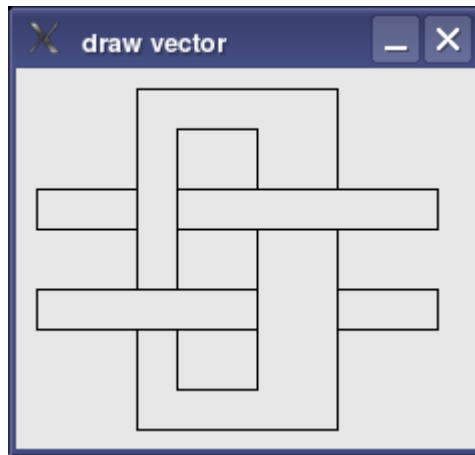
그림 12-10. 파일에 정의된 직선그리기

```
1 /* drawvector.cpp */
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include <stdio.h>
5 #include "drawvector.h"
6
7 int main(int argc, char **argv)
8 {
9      KApplication app(argc, argv,  "drawvector");
10     DrawVector drawvector;
11     drawvector.show();
12     app.setMainWidget(&drawvector);
13     return  (app.exec());
14 }
15 DrawVector::DrawVector(QWidget *parent,const
16     char *name) : QWidget(parent,name)
17 {
```

```
18     setFixedSize(230,190);
19 }
20 void DrawVector::paintEvent(QPaintEvent *)
21 {
22     FILE *fd;
23     char code[20];
24     int x;
25     int y;
26
27     if((fd = fopen("points.dat","r")) != NULL) {
28             QPainter p(this);
29             while(fscanf(fd,  "%s %d %d",code,&x,&y) == 3) {
30                     if(code[0] == 'm')
31                             p.moveTo(x,y);
32                     else if(code[0] == 'd')
33                             p.lineTo(x,y);
34             }
35             fclose(fd);
36     }
37 }
```

모든 그리기는 28～34행의 순환에서 수행된다. 28행은 QPainter객체를 만들어서 도형그리기조작을 초기화한다. 29행의 fscanf()호출은 한행의 입력자료 즉 명령, x자리표, y자리표를 읽어들인다. 명령이 현재 유표를 이동시키는것이라면 메쏘드 moveTo()는 31행에서 호출된다. 명령이 현재 유표위치로부터 새로운 위치까지 직선을 그리는것이라면 33행에서 lineTo()호출이 이루어진다.

제10절. 선분과 다각형

몇가지 QPainter메쏘드들은 QPointArray객체에 점모임을 보관하고 그다음 점들을 리용하여 다각형을 그린다. 다음 프로그램은 선분모임을 그리는 방법들중 일부를 보여준다.

```
1 /* drawpoly.cpp */
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include "drawpoly.h"
5
6 int main(int argc, char **argv)
```

```
7 {
8      KApplication app(argc, argv,  "drawpoly");
9      DrawPoly drawpoly;
10     drawpoly.show();
11     app.setMainWidget(&drawpoly);
12     return  (app.exec());
13 }
14 DrawPoly::DrawPoly(QWidget *parent,const
15     char *name) : QWidget(parent,name)
16 {
17     setFixedSize(500,100);
18 }
19 void DrawPoly::paintEvent(QPaintEvent *)
20 {
21     int offset = 0;
22     QPointArray parray(10);
23     QPainter p(this);
24
25     setPoints(parray,offset);
26     p.drawLineSegments(parray);
27
28     setPoints(parray,offset += 100);
29     p.drawPolyline(parray);
30
31     setPoints(parray,offset += 100);
32     p.drawPolygon(parray);
33
34     p.setBrush(QColor("white"));
35     setPoints(parray,offset += 100);
36     p.drawPolygon(parray,TRUE);
37
38     setPoints(parray,offset += 100);
39     p.drawPolygon(parray,FALSE);
40 }
41 void DrawPoly::setPoints(QPointArray &parray,int offset)
```

```
42 {
43     parray.setPoint(0,10+offset,50);
44     parray.setPoint(1,70+offset,50);
45     parray.setPoint(2,70+offset,30);
46     parray.setPoint(3,50+offset,30);
47     parray.setPoint(4,50+offset,90);
48     parray.setPoint(5,30+offset,90);
49     parray.setPoint(6,30+offset,10);
50     parray.setPoint(7,90+offset,10);
51     parray.setPoint(8,90+offset,70);
52     parray.setPoint(9,10+offset,70);
53 }
```

41행의 setPoints()메쏘드는 배렬에 점들을 삽입한다. 이 점모임은 그림 12-11과 같이 매 그리기에 사용되지만 이때 례외로 수평위치가 변위계산의 량만큼 오른쪽으로 이동한다.
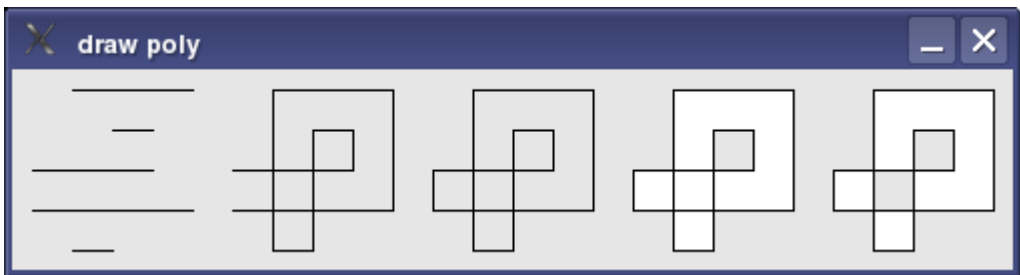
그림 12-11. 여러가지 형태의 다각형그리기

26행의 drawLineSegments()호출은 그림 12-11의 첫번째 다각형을 그린다. 이때 오직 선분만 그리므로 그것들이 결합되지 않는다. 즉 첫째 선분은 point[0]과 point[1]사이에서 그려지며 둘째는 point[2]과 point[3]사이에서 그려진다. 선분을 이루는 점들은 점들의 배렬에서 2개의 성분이여야 한다. 물론 선분의 끝점을 다음 선분의 출발점으로 사용하여 선분들을 이어 다각형을 만들수도 있다.

32행의 drawPolyLine()호출은 drawLineSegments()와 같은 입력정보를 사용하지만 그것은 앞의 선분이 끝난 점에서부터 새로운 선분을 시작함으로써 모든 선분들이 련결된 다각형을 그린다. 즉 첫째 선분은 point[0]과 point[1], 둘째 선분은 point[1]과 point[2], …으로 그려진다. 그런데 점배렬에서 마지막 점과 첫 점이 일치하지 않으므로 다각형은 닫기지 않는다.

32행의 drawPolygon()호출은 drawLineSgemetns()와 같은 방법으로 도형을 그리지만 끝점으로부터 시작점까지 반대로 선분을 그리므로 닫긴 도형을 만든다.

36행의 drawPolygon()호출은 QBrush가 QPainter객체에 보관된 후에 다각형을 그리며 그 결과다각형을 색칠한다. 다른 어떤 도형에서나 구역의 륜곽선을 그리기전에 구역을 색칠하므로 륜곽선이 색칠한 구역우에 나타나게 된다. 메쏘드호출의 둘째 인수는 겹침부분색칠규

칙을 TRUE로 설정하는데 이것은 다각형의 모든 구역들이 겹침에 관계없이 채워진다는것을 의미한다.

39행의 drawPolygon()호출은 앞의것과 동일하지만 겹침부분색칠규칙은 FALSE로 설정된다. 이 설정은 도색되는 다각형의 령역들서 한 령역이 다른 령역에 가리워진다. 그러므로 그림 12-11에서 제일 오른쪽 그림에서 도형의 겹침구역은 색칠되지 않는다는것을 보여준다. 즉 겹침점에는 두 층의 도형이 있다. 만일 도형이 3개 층으로 되여있으면 다시 색칠된다.

제11절. 타원과 원

원은 높이와 너비가 같은 타원이므로 메쏘드 drawEllipse()는 원과 타원을 그리는데 사용된다. 다음 프로그람은 그림 12-12와 같이 2개 타원과 한개 원을 가진 창문을 현시한다.

```
1 /* drawellipse.cpp */
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include "drawellipse.h"
5
6 int main(int argc, char **argv)
7 {
8     KApplication app(argc, argv,  "drawellipse");
9     DrawEllipse drawellipse;
10    drawellipse.show();
11    app.setMainWidget(&drawellipse);
12    return  (app.exec());
13 }
14 DrawEllipse::DrawEllipse(QWidget *parent,const
15    char *name) : QWidget(parent,name)
16 {
17    setFixedSize(260,140);
18 }
19 void DrawEllipse::paintEvent(QPaintEvent *)
20 {
21    QPainter p(this);
22
23    p.drawEllipse(10,50,110,40);
24    p.setBrush(QColor("white"));
```

```
25    p.drawEllipse(130,25,90,90);
26    p.setPen(NoPen);
27    p.drawEllipse(230,10,20,120);
28 }
```
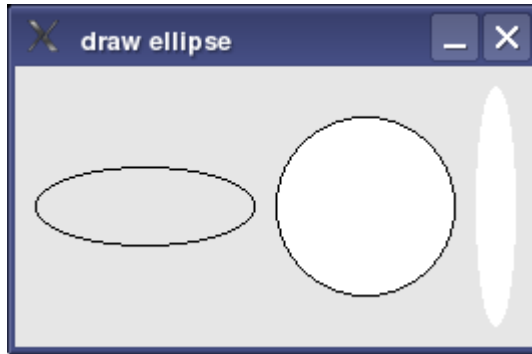
그림 12-12. 2개 타원과 한개 원

drawEllipse()메쏘드는 타원의 4개 측면을 지정하기 위하여 경계직4각형을 정의할것을 요구한다. 경계직4각형은 그 왼쪽웃구석의 x와 y자리표, 그리고 직4각형의 너비와 높이에 의해 정의된다. 실례로 그림 12-12의 왼쪽에 있는 타원은 23행의 drawEllipse()호출에 의해 그려지며 왼쪽웃구석은 왼변으로부터 10화소, 우로부터 50화소에 있다. 타원의 너비는 110화소이고 높이는 40화소이다.

QBrush객체는 24행의 setBrush()호출에 의해 QPainter에 추가되므로 나머지 타원들은 솔색으로 칠해진다. 26행은 setPen()을 호출하여 QPainter로부터 펜을 삭제하므로 오른쪽 타원은 륜곽선을 가지지 않는다.

왼쪽웃구석대신에 중심점주위에 원이나 타원을 그릴 필요가 있다. 그러기 위해서는 간단히 중심점으로부터 매개 방향으로 반경을 덜어서 왼쪽웃구석의 위치를 지정한다.

```
p.drawEllipse(x - (w / 2),y - (h / 2),w,h);
```

제12절. 원과 타원의 부분그리기

원이나 타원의 일부분만을 그리는 방법이 3가지 있다. 그 과정은 앞의 실례와 같이 원 또는 타원을 그리는것과 같지만 시작각도와 마감각도를 지정해야 한다.

각도는 도의 16분의 1단위로 계산된다. 고정코드의 각도를 입력하려고 하는 경우에 일반적으로 사용하는 각도에 대하여 표 12-2에 제시한다.

표 12-2. 각도계산단위들의 비교

Qt단위	도	rad
0	0	0
720	45	0.7854
1440	90	1.5708

Qt단위	도	rad
2160	135	2.3562
2880	180	3.1416
3600	225	3.9270
4320	270	4.7124
5040	315	5.4978
5760	360	6.2832

각도를 계산하려고 한다면 대부분의 수학프로그람들은 도 또는 라디안을 사용하므로 그것들을 변환하여야 한다. 다음 명령문은 도와 라디안을 Qt비률로 변환한다.

```
angle = degree * 16;
angle = (radian * 180) / PI;
```

반대로 Qt비률을 도와 라디안으로 변환하기 위해서는 다음의 명령문을 리용한다.

```
degree = angle / 16;
radian = (angle * PI) / 180;
```

각도의 정방향회전은 시계바늘과 반대방향이다. 0°점은 오른쪽으로 한다. 시작각도는 그리기가 시작하는 0점으로부터 거리를 지정하며 마감각도는 시작각도로부터 그리기의 끝까지 거리를 지정한다. 두 수값은 정수 혹은 부수가 될수 있다. 시작각도가 마감각도보다 크면 그리기는 정방향(시계바늘과 반대방향)으로 진행되며 시작각도가 마감각도보다 작으면 그리기는 역방향(시계바늘방향)으로 진행된다.

다음 실례는 호를 그리는 3가지 수법을 설명한다.

```
1 /* arcpiechord.cpp */
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include "arcpiechord.h"
5
6 int main(int argc, char **argv)
7 {
8      KApplication app(argc, argv,  "arcpiechord");
9      ArcPieChord arcpiechord;
10     arcpiechord.show();
11     app.setMainWidget(&arcpiechord);
12     return  (app.exec());
13 }
14 ArcPieChord::ArcPieChord(QWidget *parent,const
15     char *name) : QWidget(parent,name)
```

```
16 {
17    setFixedSize(260,420);
18 }
19 void ArcPieChord::paintEvent(QPaintEvent *)
20 {
21    QPainter p(this);
22
23    p.drawArc(10,50,110,40,0,4000);
24    p.drawChord(10,190,110,40,0,4000);
25    p.drawPie(10,330,110,40,0,4000);
26    p.setBrush(QColor("white"));
27    p.drawArc(130,25,90,90,0,2000);
28    p.drawChord(130,165,90,90,0,2000);
29    p.drawPie(130,305,90,90,0,2000);
30    p.setPen(NoPen);
31    p.drawArc(230,10,20,120,720,4320);
32    p.drawChord(230,150,20,120,720,4320);
33    p.drawPie(230,290,20,120,720,4320);
34 }
```

23행의 drawArc()호출은 그림 12-13의 왼쪽웃구석에 도형을 만든다. 이 도형은 흑색펜과 솔이 없는 기정QPainter를 사용하여 그려지는데 타원과 같은 경계직4각형수법을 리용한다. 즉 타원의 일부분만 그리지만 전체 타원의 경계직4각형의 왼쪽웃구석의 x와 y를 선택한다. 시작각도는 0, 마감각도는 약 270도인 4000이다.
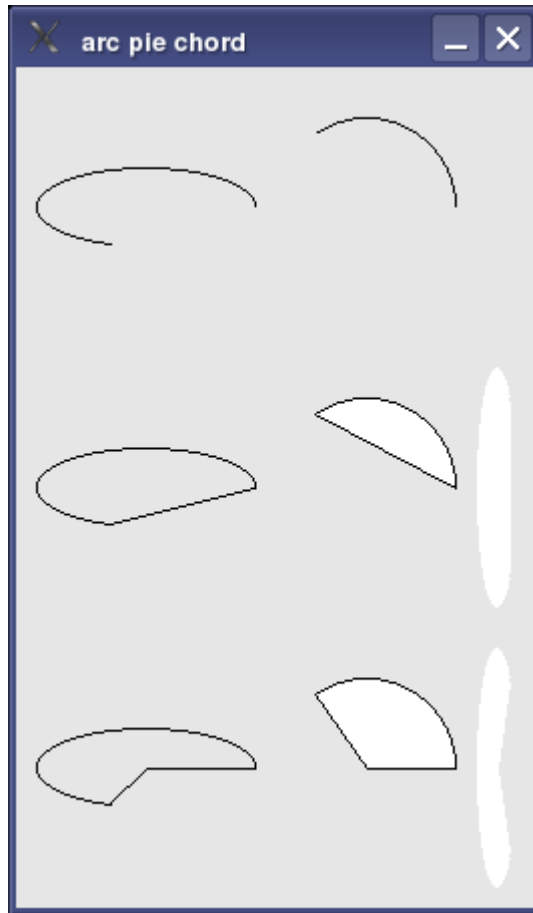
그림 12-13. 호, 현, 부채형을 그리는 방법

27행의 drawArc()호출은 그림 12-13의 첫행의 중심에 도형을 그린다. 이 도형은 솔을 가지고있는 QPainter에 의해 그려지지만 호가 닫긴 도형이 아니므로 색칠되지 않는다. 31행의 drawArc()호출은 펜이 금지되였고 drawArc()가 솔을 사용하지 않으므로 나타나지 않는다.

24행의 drawChord()호출은 그림 12-13의 중심행의 제일 왼쪽에 그린다. 현은 호의 끝점들사이에 직선을 그어서 닫긴 도형을 만드는것을 제외하면 호와 같다. 현이 닫긴 도형이므로 28행과 32행의 drawChord()호출은 둘다 닫긴 구역을 솔색으로 칠한다.

25행의 drawPie()호출은 그림 12-13의 바닥행의 제일 왼쪽 도형을 그린다. 부채형은 중심과 2개 끝점사이에 2개 직선을 그어서 닫긴 도형을 만드는것외에는 호와 같다. 부처형은 닫긴 도형이므로 29행과 33행의 drawPie()호출은 모두 닫긴 구역을 솔색으로 칠한다.

제13절. 환4각형

QPainter의 메쏘드 drawRoundRect()는 환4각형을 그리는데 사용된다. 다음 실례는 바른4각형, 직4각형, 원, 그리고 타원과 함께 환4각형을 그리는데 사용할수 있는 drawRoundRect()의 유연성을 보여준다. 프로그람은 그림 12-14와 같이 많은 도형을 그린다.

```
1 /* roundrectangle.cpp */
```

```
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include "roundrectangle.h"
5
6 int main(int argc, char **argv)
7 {
8     KApplication app(argc, argv,  "roundrectangle");
9     RoundRectangle roundrectangle;
10    roundrectangle.show();
11    app.setMainWidget(&roundrectangle);
12    return  (app.exec());
13 }
14 RoundRectangle::RoundRectangle(QWidget *parent,const
15    char *name) : QWidget(parent,name)
16 {
17    setFixedSize(190,370);
18 }
19 void RoundRectangle::paintEvent(QPaintEvent *)
20 {
21    QPainter p(this);
22    p.setBrush(QColor("white"));
23
24    p.drawRoundRect(10,10,50,50);
25    p.drawText(30,35,  "1");
26
27    p.drawRoundRect(70,10,50,50,50,50);
28    p.drawText(90,35,  "2");
29
30    p.drawRoundRect(130,10,50,50,100,100);
31    p.drawText(150,35,  "3");
32
33    p.drawRoundRect(10,70,170,50);
34    p.drawText(90,95,  "4");
35
36    p.drawRoundRect(10,130,170,50,0,50);
```

```
37    p.drawText(90,155,  "5");
38
39    p.drawRoundRect(10,190,170,50,50,80);
40    p.drawText(90,215,  "6");
41
42    p.drawRoundRect(10,250,170,50,100,100);
43    p.drawText(90,275, "7");
44
45    p.drawRoundRect(10,310,170,50,9,30);
46    p.drawText(90,335, "8");
47 }
```
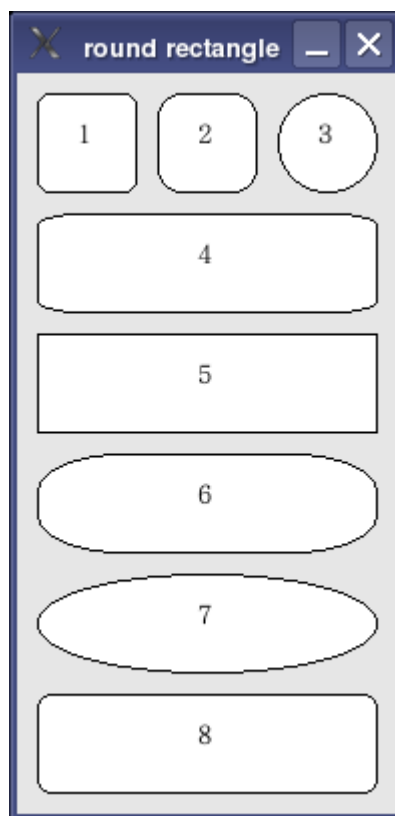
그림 12-14. 여러가지 형식의 환4각형들

다음 2가지 메쏘드들중 하나를 호출하여 환4각형들을 그린다.

drawRoundRect(int x,int y,int w,int h)

drawRoundRect(int x,int y,int w,int h, int xround,int yround)

처음 4개 인수는 직4각형을 정의한다. 마지막 2개 인수(둘다 기정값 25)는 수직과 및 수평방향에서 모서리들의 반경을 지정한다.

그림 12-14에서 직4각형 1은 24행의 drawRoundeRect()호출에 의해 얻어진다. 처음 2개

인수는 직4각형의 모서리가 원형으로 되지 않은 경우의 왼쪽웃구석의 x와 y위치를 지정한다. 도형은 한 변이 50화소인 바른4각형이고 모서리들의 반경은 x와 y방향으로 둘다 기정으로 25로 설정된다. 이것은 수직거리의 25%와 수평거리의 25%는 원형모서리들을 만드는데 사용된다는것을 의미한다.

직4각형 2는 27행의 drawRoundedRect()호출에 의해 얻어진다. 직4각형 1처럼 이 호출도 바른4각형을 생성하지만 수평과 수직반경들은 각각 기정으로 25%로 설정되지 않고 50%로 설정되였다.

직4각형 3은 높이와 너비를 같은 값으로 설정하고 모서리의 반경을 100%로 설정하면 결과가 원으로 된다는것을 보여준다.

직4각형 4는 33행에서 그려진다. 수직과 수평반경은 기정으로 25%로 허용되지만 직4각형이 높이에 비하여 너비가 더 넓으므로 보다 수평방향에 수직방향보다 더 많은 화소들이 포함되며 대칭이 아닌 곡선이 그려진다.

36행에서 그려지는 직4각형5는 모서리의 반경중 하나 혹은 둘을 0%로 설정하면 바른4각형의 모서리로 된다는것을 보여준다. 이 실례에서 모서리의 수직반경은 50%로 설정되였으나 수평반경이 0%이므로 모서리가 곡선이 될수 없다.

39행에서 만들어지는 직4각형 6은 모서리의 수직반경 100%, 수평반경 30%로 설정된다.

42행에서 그리는 직4각형 7은 모서리의 수직과 수평반경이 모두 100%로 설정되였으므로 결과는 타원이다.

45행에서 그리는 직4각형 8은 모서리가 대칭반경을 가지도록 설계된다. 즉 수직과 수평방향 모두의 곡선에 같은 화소수가 포함된다. 모서리의 반경들이 퍼센트로 표시되므로 화소값을 선택한 다음 그것을 리용하여 매개 방향에서 퍼센트를 결정하여야 한다.

```
xround = (100 * pixels) / height;
yround = (100 * pixels) / width;
```

제14절. 픽스매프와 본문의 그리기

픽스매프의 전부 또는 일부분을 그릴수 있으며 본문을 그리는데 사용할 서체를 정의할 수 있다. 다음 실례는 그림 12-15와 같이 전체 픽스매프를 그리고 그 다음에 픽스매프의 부분을 그리며 다음으로 그림의 우에 본문을 쓴다.
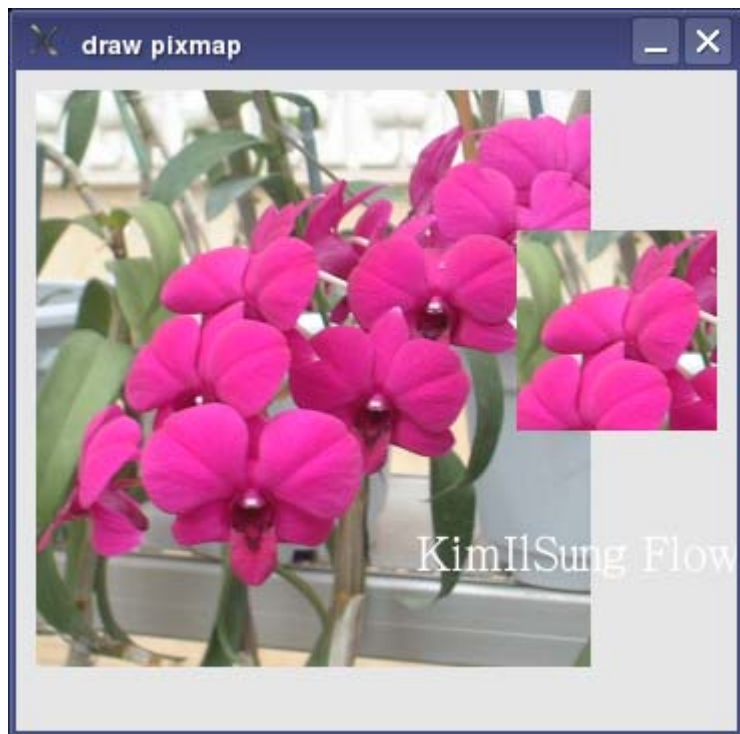
그림 12-15. 본문을 가지는 픽스매프

DrawPixmap머리부파일

```
1 /* drawpixmap.h */
2 #ifndef DRAWPIXMAP_H
3 #define DRAWPIXMAP_H
4
5 #include <qwidget.h>
6
7 class DrawPixmap: public QWidget
8 {
9 public:
10     DrawPixmap(QWidget *parent=0,const char *name=0);
11 private:
12     QPixmap logo;
13 protected:
14     virtual void paintEvent(QPaintEvent *);
15 };
16
17 #endif
```

그리려는 QPixmap는 자료로부터 오직 한번 만들어진다. 그것은 12행에서 줄임화상으로서 보관되므로 후에 현시에 사용할수 있다.

DrawPixmap

```
1 /* drawpixmap.cpp */
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include <qfont.h>
5 #include "drawpixmap.h"
6
7 #include "logo.xpm"
8
9 int main(int argc, char **argv)
10 {
11    KApplication app(argc, argv,  "drawpixmap");
12    DrawPixmap drawpixmap;
13    drawpixmap.show();
14    app.setMainWidget(&drawpixmap);
15    return  (app.exec());
16 }
17 DrawPixmap::DrawPixmap(QWidget *parent,const
18    char *name) : QWidget(parent,name)
19 {
20    logo = QPixmap(magick);
21    setFixedSize(360,330);
22 }
23 void DrawPixmap::paintEvent(QPaintEvent *)
24 {
25    QPainter p(this);
26
27    p.drawPixmap(10,10,logo);
28    p.drawPixmap(250,80,logo,50,50,100,100);
29
30    QFont font = p.font();
31    font.setPointSize(18);
32    p.setFont(font);
```

```
33
34    p.setPen(QColor("white"));
35    p.drawText(200,250, "Penguin");
36 }
```

17행에서 시작하는 구성자는 7행에서 포함된 자료파일 logo.xpm로부터 줄임화상픽스매프를 만든다. 그다음에 고정크기로 현시창문을 설정한다.

27행의 drawPixmap()호출은 전체 줄임화상픽스매프를 그린다. 픽스매프의 왼쪽웃구석은 창문부품들의 왼쪽면으로부터 10화소, 우에서부터 10화소 떨어진곳에 위치한다. 다른 인수는 지정되지 않았으므로 전체 픽스매프는 목표위치에 복사된다.

28행의 drawPixmap()호출은 줄임화상픽스매프의 한부분만 그린다. 이 메쏘드는 우선 픽스매프로부터 직4각형구역을 읽은 다음 목표창문에 그것을 그린다. 마지막 4개 메쏘드인수들은 왼쪽웃구석과 높이와 너비를 지정하고 도형을 배치할 구역을 결정한다. 배치하려는 구역은 픽스매프의 왼쪽으로부터 60화소, 우로부터 50화소이며 그 너비는 100화소, 높이는 80화소이다. 처음 2개 인수는 픽스매프를 그리려는 곳을 지정하며 그 왼쪽웃구석은 왼쪽에서 250화소, 우로부터 80화소에 배치된다.

모든 QPainter객체는 본문을 그리는데 사용하는 QFont객체를 포함한다. 기정서체를 사용하거나 새로운 서체를 만들기 또는 이 실례처럼 현존서체를 수정할수 있다. 30행의 font()호출은 QPainter객체로부터 QFont객체를 얻는다. 이 실례에서 31행의 setPointSize()호출은 본문을 좀 크게 만든다. setFont()호출은 QPainter본문을 모두 그리는데 사용되는 새로운 서체를 설정한다.

34행의 setPen()호출은 본문이 기정의 흑색대신 백색으로 나타나게 하고 35행의 drawText()호출은 창문의 왼쪽으로부터 200화소, 우로부터 250화소되는 위치에 본문을 그린다.

요 약

이 장에서 설명한 QPainter메쏘드들은 언제나 요구되는 도형처리의 90%이상 제공한다. 도형그리는 2개의 객체 QPen과 QBrush만을 리용하여 모든 도형을 그릴수 있다. 특수한 경우에 필요한것을 정확히 얻기 위하여 화소별수법을 사용할수 있다.

이 장은 다음과 같은 그리기를 하는데 사용할수 있는 QPainter메쏘드들을 설명하였다.

- 창문에 한번에 한 화소 그리거나 혹은 화소배렬을 보관할 객체들을 정의하고 그것들을 한번에 그린다.
- 한 점에서 다른 점까지 여러색과 여러 크기로 직선을 그린다. 또한 여러 선분을 한번에 하나씩 또는 모두 그릴수 있다.
- 타원과 원을 모두 그리거나 그것들의 일부분만을 그릴수 있다. 각이한 형식으로 원과 타원들을 색칠하고 잘라낼수 있다.
- 픽스매프들을 전부 그리거나 직4각형구역을 선택하여 그린다.

제13장. 도형에 대한 조작

학습내용

도형조작명령렬의 보관 및 현시기에 재생하기

창문과 다른 도형들의 인쇄

인쇄기의 도형그리기능력의 질문

도형의 비례확대축소, 오려내기, 경사지기, 회전 및 변환

그려낸 도형과 화상의 렬로 동화만들기

비트준위에서 화소색들의 조작

앞 장은 창문에 도형을 그리고 색칠하는 기초를 설명하였으나 이 장은 KDE와 Qt에서 도형조작을 위한 특별한 몇가지 기능들을 설명한다.

이 장에서 설명한 대부분의 기술은 도형의 내용을 변경하는데 사용될수 있다. 아마도 가장 유용한 정보에는 화상들의 회전, 배치를 합리적으로 진행하기 위한 내용들이 더 많이 포함되여야 한다. 색 또는 흑백색으로 도형화상을 인쇄하는것은 매우 간단한 과정이다. 그러나 도형을 비례확대축소하고 경사지게 하여 모양을 변경할수 있으며 지어 매 화소의 비트값들을 변경하여 수정할수 있고 동화는 한개 프레임씩 그리고 시간조종렬로 프레임들을 현시할수 있다.

제1절. QPicture에 의한 도형의 보관

창문부품의 창문에 그릴수 있는것은 QPicture객체에 그릴수 있다. QPicture객체는 디스크파일에 그리기명령을 보관할수 있고 다른 QPicture객체는 파일을 읽어들이고 그 그리기지령을 실행할수 있다. 복잡한 그림을 보관하고 한 체계에서 다른 체계에로 도형을 전송하는것을 비롯하여 도형을 리용하는 목적은 많다. 다음 프로그람은 간단한 그림을 만들고 디스크파일에 보관한다.

Record

```
1 /* record.cpp */
2 #include <iostream.h>
3 #include <kapplication.h>
4 #include <qpainter.h>
5 #include <qpicture.h>
6 #include <qwidget.h>
7
8 int main(int argc,char **argv)
9 {
10    KApplication app(argc,argv,  "record");
```

```
11    QPainter paint;
12    QPicture pic;
13
14    paint.begin(&pic);
15    paint.setBrush(QColor("black"));
16    paint.drawRect(50,75,350,100);
17    paint.setBrush(QColor("white"));
18    paint.drawEllipse(150,50,150,150);
19    paint.setPen(QWidget::NoPen);
20    paint.drawRect(100,100,250,50);
21    paint.end();
22    if(!pic.save("recplay.qpic"))
23          cout << "Unable to create recplay.qpic" << endl;
24 }
```

이 프로그람은 도형을 만들지만 창문을 현시하지 않는다. 대신에 그림에서 QPicture객체를 리용하며 QPicture객체는 모든 지령을 기록한 다음 디스크파일에 써넣는다.

10행에서 KApplication객체 app는 QPainter객체가 오직 KDE응용프로그람에서만 사용될수 있으므로 KDE응용프로그람에서처럼 이것을 정의한다. 11행과 12행은 그리기에서 사용되는 QPainter객체와 QPainter지령을 기록하는 QPicture객체를 만든다.

14행은 begin()을 호출함으로써 도형그리기를 위한 대화를 시작한다. 그리기객체는 창문부품이 아니라 QPicture객체이다. 15～20행은 QPainter 펜과 솔값을 설정하고 실제로 메쏘드를 호출하여 그린다. QPicture객체는 매 메쏘드호출들을 보존한다. 그리기대화는 21행의 end()호출에 의해 정지된다. 22행의 save()호출은 recplay라는 파일을 만든다. qpic는 모든 그리기명령들을 가지고있다.

Playback머리부파일

```
1 /* playback.h */
2 #ifndef PLAYBACK_H
3 #define PLAYBACK_H
4
5 #include <qwidget.h>
6
7 class Playback: public QWidget
8 {
9 public:
10    Playback(QWidget *parent=0,const char *name=0);
```

```
11 protected:
12     virtual void paintEvent(QPaintEvent *);
13 };
14
15 #endif
```

Playback

```
1 /* playback.cpp */
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include <qpicture.h>
5 #include "playback.h"
6
7 int main(int argc,char **argv)
8 {
9     KApplication app(argc,argv,  "playback");
10    Playback playback;
11    playback.show();
12    app.setMainWidget(&playback);
13    return  (app.exec());
14 }
15 Playback::Playback(QWidget *parent,const
16    char *name) : QWidget(parent,name)
17 {
18    setFixedSize(450,250);
19 }
20 void Playback::paintEvent(QPaintEvent *)
21 {
22    QPainter p(this);
23    QPicture picture;
24
25    if(picture.load("recplay.qpic"))
26            p.drawPicture(picture);
27 }
```

20행의 paintEvent()메쏘드는 보관된 명령을 얻기 위해 QPicture객체를 만든다. 25행의 load()호출은 명령목록을 얻고 load()호출이 성공하면 26행의 drawPicture()호출은 파일에 보관

된 모든 명령을 실행한다. 결과는 그림 13-1에 보여준 창문이다.
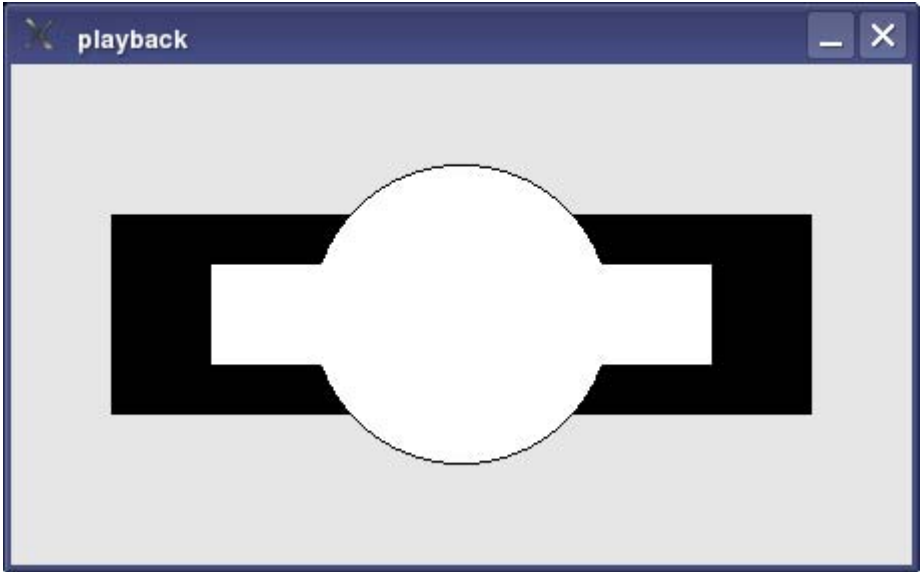
그림 13-1. 이전에 보관한 도형그리기명령의 재생

이미 보관된 도형그리기명령들의 재생은 QPainter객체를 사용하므로 프로그람이 보관된 명령들을 리용하는데서 문제가 발생하지 않는다. 실례로 그림 13-2에 보여주는 화상은 paintEvent()메쏘드의 paint명령들을 다음과 같이 변경한 결과이다.

```
if(picture.load("recplay.qpic"))
        p.drawPicture(picture);
p.setBrush(QColor("black"));
p.drawRect(110,110,230,30);
```
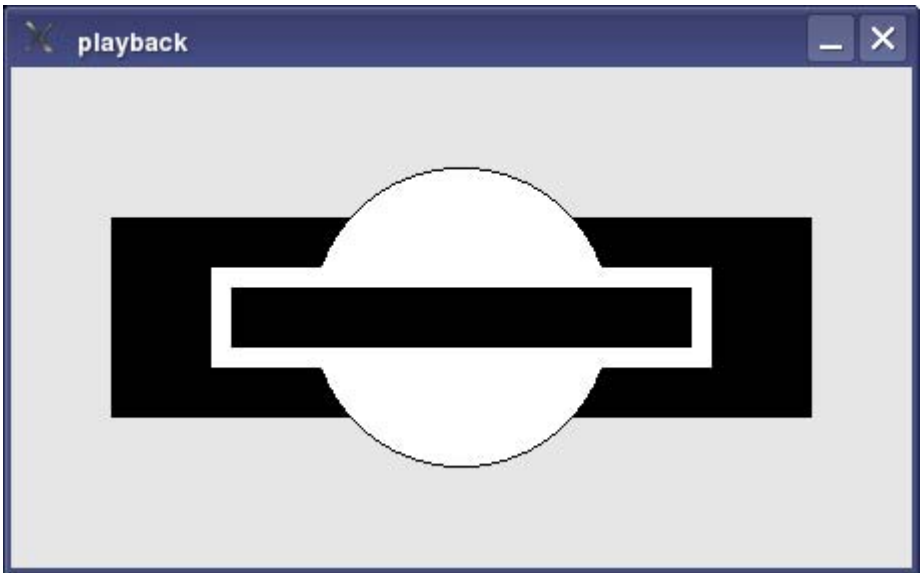
그림 13-2. 재생과 현재 도형명령의 결합

제2절. 인쇄기에 도형의 그리기

인쇄기에 대한 화면인쇄는 현시기에 창문을 그리는것처럼 간단하다. 다음 실례프로그람은 이전에 그림 13-1에 보여준것과 같은 도형창문을 현시한다. 다른것은 오른쪽아래구석에 Print단추를 추가한것이다. 단추선택은 그림 13-3에 보여준 대화칸을 펼치면 인쇄에 대한 설정을 할수 있다. 사용자가 OK단추를 찰칵하면 도형이 인쇄된다.

PrintGraphic머리부파일

```
1 /* printgraphic.h */
2 #ifndef PRINTGRAPHIC_H
3 #define PRINTGRAPHIC_H
4
5 #include <qwidget.h>
6 #include <qpushbutton.h>
7
8 class PrintGraphic: public QWidget
9 {
10     Q_OBJECT
11 public:
12     PrintGraphic(QWidget *parent=0,const char *name=0);
13 private:
14     QPushButton *printButton;
15 private slots:
16     void printSlot();
17 protected:
18     virtual void paintEvent(QPaintEvent *);
19 };
20
21 #endif
```

PrintGraphic

```
1 /* printgraphic.cpp */
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include <qprinter.h>
5 #include "printgraphic.h"
6
```

```
7 int main(int argc,char **argv)
8 {
9     KApplication app(argc,argv,  "printgraphic");
10    PrintGraphic printgraphic;
11    printgraphic.show();
12    app.setMainWidget(&printgraphic);
13    return(app.exec());
14 }
15 PrintGraphic::PrintGraphic(QWidget *parent,const
16    char *name) : QWidget(parent,name)
17 {
18    setFixedSize(450,250);
19    printButton = new QPushButton("Print",this);
20    printButton->setGeometry(370,200,70,40);
21    connect(printButton,SIGNAL(clicked()),
22            this,SLOT(printSlot()));
23 }
24 void PrintGraphic::paintEvent(QPaintEvent *)
25 {
26    QPainter paint;
27
28    paint.begin(this);
29    paint.setBrush(QColor("black"));
30    paint.drawRect(50,75,350,100);
31    paint.setBrush(QColor("white"));
32    paint.drawEllipse(150,50,150,150);
33    paint.setPen(QWidget::NoPen);
34    paint.drawRect(100,100,250,50);
35    paint.end();
36 }
37 void PrintGraphic::printSlot()
38 {
39    QPainter paint;
40    QPrinter printer;
41
```

```
42     if(printer.setup(this)) {
43             paint.begin(&printer);
44             paint.setBrush(QColor("black"));
45             paint.drawRect(50,75,350,100);
46             paint.setBrush(QColor("white"));
47             paint.drawEllipse(150,50,150,150);
48             paint.setPen(QWidget::NoPen);
49             paint.drawRect(100,100,250,50);
50             paint.end();
51     }
52 }
```
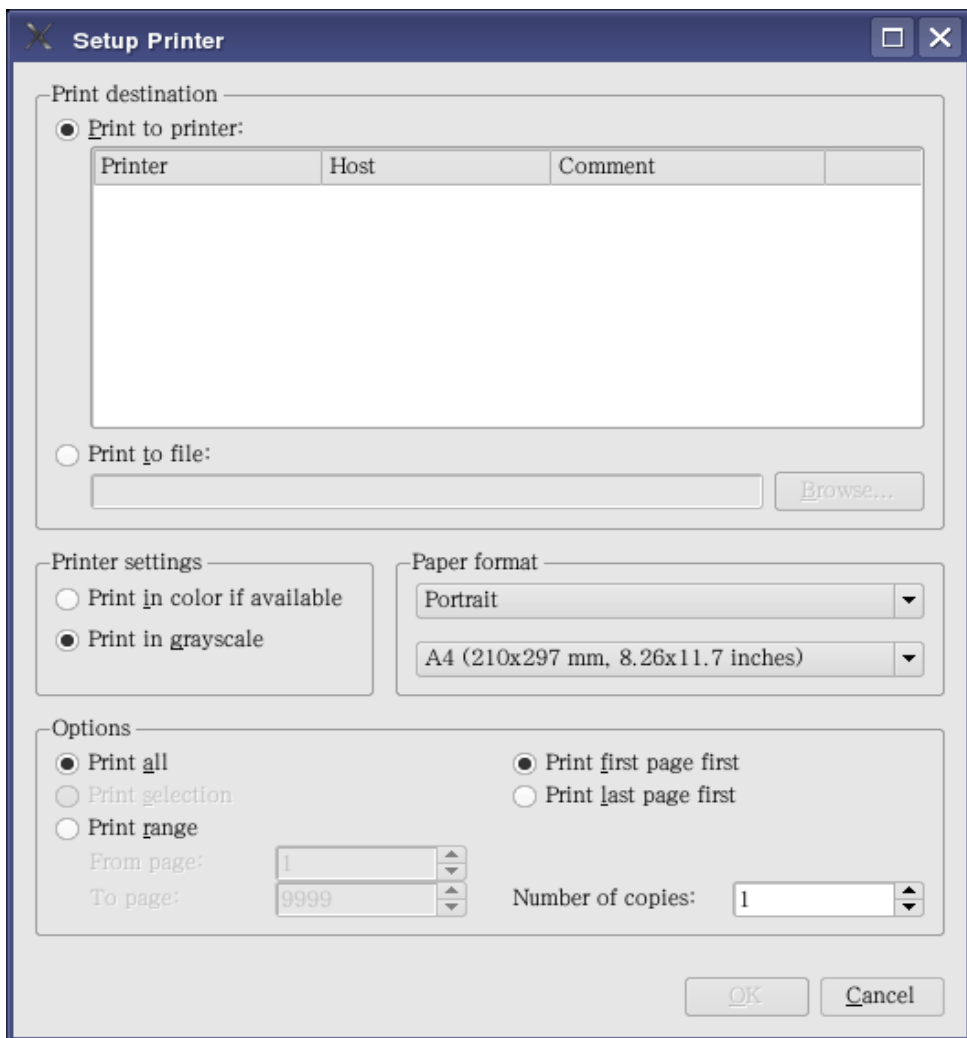
그림 13-3. 인쇄설정대화칸

15행의 구성자는 창문크기를 설정하고 오른쪽아래구석에 단추를 설치한다. 처리부메쏘드 printSlot()는 단추에 련결된다.

24행의 paintEvent()메쏘드는 창문부품의 창문에 도형을 그린다.

37행의 처리부메쏘드 printSlot()는 사용자에게 인쇄기환경설정을 의뢰하고 사용자가 그림 13-3에 보여준 대화칸에서 OK단추를 찰칵하면 인쇄기로 도형을 그린다. 42행의 setup()호출은 대화칸을 펼치고 이때 TRUE의 돌림값은 인쇄가 수행된다는것을 가리킨다. 43행의 begin()호출은 QPainter객체를 인쇄기에 련결한다.

50행의 end()호출은 그리기를 끝내고 인쇄기에 도형명령을 보낸다. 이 호출은 또한 인쇄기에 대한 출력을 완료하고 인쇄용 완충기에 페지(또는 페지들)를 보낸다. 인쇄도중에 현재 페지를 취소하고 새 페지를 인쇄하려고 한다면 다음 메쏘드를 호출하면 된다.

```
print.newPage();
```

인쇄과정의 임의의 시점에서 완충기에 인쇄하려는 모든 페지를 보내지기전에 다음과 같이 메쏘드를 호출하면 이미 보낸 페지들을 삭제할수 있다.

```
print.abort ( ) ;
```

제3절. 인쇄기정보와 조종

창문에 그리는것처럼 인쇄기로 인쇄하는것은 간단하지만 페지크기와 인치당 점의 개수와 같은 정보를 알 필요가 있다. 다음 실례프로그람은 그림 13-4와 같이 기본인쇄기정보를 현시한다. 이 프로그람을 실행하고 대화칸을 펼치여 인쇄기에 대한 환경설정을 진행할수 있다.
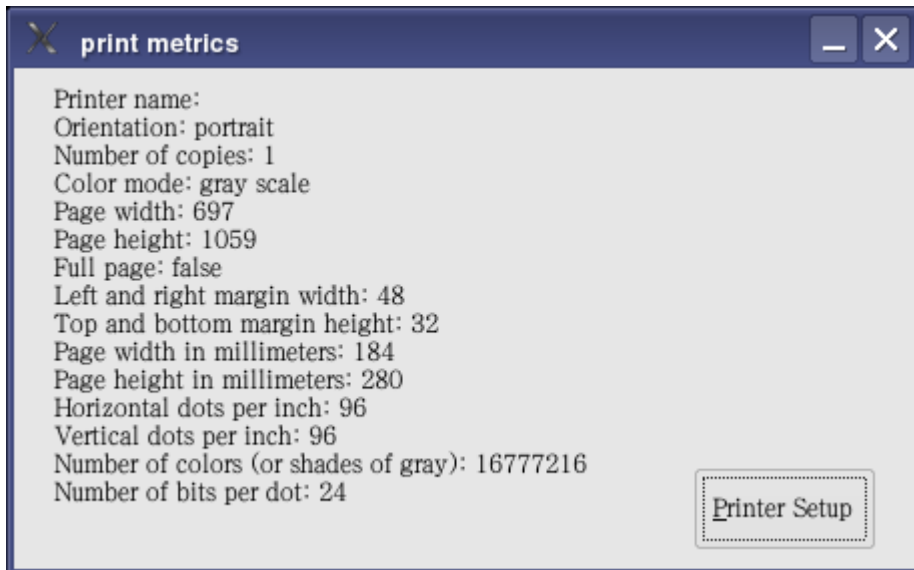
그림 13-4. 인쇄정보대화칸

PrintMetrics머리부파일

```
1 /* printmetrics.h */
```

```
2 #ifndef PRINTMETRICS_H
3 #define PRINTMETRICS_H
4
5 #include <qwidget.h>
6 #include <qpushbutton.h>
7 #include <qprinter.h>
8
9 class PrintMetrics: public QWidget
10 {
11     Q_OBJECT
12 public:
13     PrintMetrics(QWidget *parent=0,const char *name=0);
14 private:
15     QPrinter printer;
16     QPushButton *printButton;
17 private slots:
18     void printSetupSlot();
19 protected:
20     virtual void paintEvent(QPaintEvent *);
21 };
22
23 #endif
```

QPrinter객체는 클라스의 부분으로 포함된다. 그것은 클라스의 성원이므로 사용자에 의한 어떠한 환경변화도 PrintMetrics객체의 수명에 영향을 주지 않는다.

PrintMetrics

```
1 /* printmetrics.cpp */
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include <qpaintdevicemetrics.h>
5 #include "printmetrics.h"
6
7 int main(int argc,char **argv)
8 {
9     KApplication app(argc,argv,  "printmetrics");
10    PrintMetrics printmetrics;
```

```
11    printmetrics.show();
12    app.setMainWidget(&printmetrics);
13    return  (app.exec());
14 }
15 PrintMetrics::PrintMetrics(QWidget *parent,const
16    char *name) : QWidget(parent,name)
17 {
18    setFixedSize(450,250);
19    printButton = new QPushButton("Printer Setup",this);
20    printButton->setGeometry(340,200,90,40);
21    connect(printButton,SIGNAL(clicked()),
22            this,SLOT(printSetupSlot()));
23 }
24 void PrintMetrics::printSetupSlot()
25 {
26    printer.setup(this);
27 }
28 void PrintMetrics::paintEvent(QPaintEvent *)
29 {
30    QPainter paint;
31    QPaintDeviceMetrics metrics(&printer);
32    QString string;
33
34    paint.begin(this);
35
36    QFontMetrics fm = paint.fontMetrics();
37    int x = 20;
38    int y = 20;
39
40    string = "Printer name: " + printer.printerName();
41    paint.drawText(x,y,string);
42    y += fm.height();
43
44    if(printer.outputToFile()) {
45            string = "Output to file: "
```

```
46                    + printer.outputFileName();
47          paint.drawText(x,y,string);
48          y += fm.height();
49  }
50
51  if(printer.orientation() == QPrinter::Portrait)
52          string = "Orientation: portrait";
53  else
54          string = "Orientation: landscape";
55  paint.drawText(x,y,string);
56  y += fm.height();
57
58  string = "Number of copies: ";
59  string += QString::number(printer.numCopies());
60  paint.drawText(x,y,string);
61  y += fm.height();
62
63  if(printer.colorMode() == QPrinter::GrayScale)
64          string = "Color mode: gray scale";
65  else
66          string = "Color mode: color";
67  paint.drawText(x,y,string);
68  y += fm.height();
69
70  string = "Page width: ";
71  string += QString::number(metrics.width());
72  paint.drawText(x,y,string);
73  y += fm.height();
74
75  string = "Page height: ";
76  string += QString::number(metrics.height());
77  paint.drawText(x,y,string);
78  y += fm.height();
79
80  if(printer.fullPage())
```

```
81          string = "Full page: true";
82      else
83          string = "Full page: false";
84      paint.drawText(x,y,string);
85      y += fm.height();
86
87      string = "Left and right margin width: ";
88      string += QString::number(printer.margins().width());
89      paint.drawText(x,y,string);
90      y += fm.height();
91
92      string = "Top and bottom margin height: ";
93      string += QString::number(printer.margins().height());
94      paint.drawText(x,y,string);
95      y += fm.height();
96
97      string = "Page width in millimeters: ";
98      string += QString::number(metrics.widthMM());
99      paint.drawText(x,y,string);
100     y += fm.height();
101
102     string = "Page height in millimeters: ";
103     string += QString::number(metrics.heightMM());
104     paint.drawText(x,y,string);
105     y += fm.height();
106
107     string = "Horizontal dots per inch: ";
108     string += QString::number(metrics.logicalDpiX());
109     paint.drawText(x,y,string);
110     y += fm.height();
111
112     string = "Vertical dots per inch: ";
113     string += QString::number(metrics.logicalDpiY());
114     paint.drawText(x,y,string);
115     y += fm.height();
```

```
116
117   string = "Number of colors (or shades of gray): ";
118   string += QString::number(metrics.numColors());
119   paint.drawText(x,y,string);
120   y += fm.height();
121
122   string = "Number of bits per dot: ";
123   string += QString::number(metrics.depth());
124   paint.drawText(x,y,string);
125   y += fm.height();
126
127   paint.end();
128 }
```

24행의 처리부메쏘드 printSetupSlot()는 그림 13-3과 같이 인쇄기환경설정대화칸을 펼친다. 이것은 인쇄설정을 변경하는데 쓰인다.

28행의 paintEvent()메쏘드는 현재 인쇄기설명정보와 환경의 목록을 현시하는 창문을 만든다. 일부 정보는 QPaintDeviceMetrics객체로부터 얻고 일부는 QPrinter객체자체로부터 직접 얻는다.

36행의 인쇄기이름은 사용자가 사용가능한 인쇄기들의 목록으로부터 선택한 이름이다. 사용자가 인쇄기를 선택하지 않았다면 빈문자렬이 printerName()에 의해 되돌려지며 인쇄출력은 기정인쇄기에 대하여 진행된다. 출력이 기정인쇄기대신 파일에 대하여 진행되면 44행의 outputToFile()호출은 TRUE를 되돌려주며 outputFileName()호출은 파일이름을 되돌려준다. 파일이 인쇄될 때 인쇄된 자료는 postscript형식으로 파일에 보내진다.

51행의 orientation()호출은 자료가 세로방식으로 인쇄되는가 가로방식으로 인쇄되는가 가리킨다. 기정방식은 세로방식이다.

59행의 numCopies()호출은 인쇄할 문서수를 되돌려주며 기정값은 1이다.

63행의 colorMode()호출은 인쇄할 때 방식을 알려준다.

71행과 76행은 인쇄가능구역의 페지너비와 높이를 지정한다. 도형을 표시하기 위한 높이와 너비값은 필요하면 창문에 도형을 그릴 때처럼 이 수를 여백의 값들과 전페지지시기와 결합하여 사용할수 있다. 80행의 fullPage()호출이 TRUE를 되돌려준다면 높이와 너비은 종이의 경계까지 늘어난다. fullPage()가 FALSE를 되돌려준다면 높이와 너비는 여백내의 치수이다. printer.setFullPage(TRUE)를 호출하여 인쇄기를 완전페지방식으로 설정할수 있다. 또는 printer.setFullPage(FALSE)를 호출함으로써 페지크기를 여백(기정)으로 조절되도록 설정할수 있다.

88행과 93행에서 margins()호출에 의해 얻은 여백값은 전페지방식이 아닐 때 실제여백

값이고 전페지방식일 때 제안된 여백이다.

고정너비와 높이를 가지지 않는 창문과 달리 인쇄기는 고정적인 물리너비와 인치당 점의 개수를 가지므로 이 값들을 결정할수 있다. 그러나 인쇄기가 콤퓨터에 값들을 통보하지 않을수 있으므로 수값들을 전적으로 믿을수는 없으며 또한 인쇄기환경이 잘못되거나 사용자가 파일을 출력하려고 선택하였을수 있다. 어쨌든 페지의 너비와 높이는 98행과 103행에서 mm로 통보되며 인치당 점의 개수는 108행과 113행에서 통보된다.

점당 총색수(또는 회색그늘수)는 118행의 numColors()호출로부터 얻는다. 같은 정보는 또한 123행의 depth()호출에서 매 점의 비트수로 통보된다.

제4절. 창문에 그림을 맞추기

자체의 자리표계를 설정하고 그것을 리용하여 창문에 도형을 그리고 자리표계를 실제 창문의 자리표계로 자동적으로 변환할수 있다. 다음 실례는 그 방법을 보여준다.

FitWindow머리부파일

```
1 /* fitwindow.h */
2 #ifndef FITWINDOW_H
3 #define FITWINDOW_H
4
5 #include <qwidget.h>
6
7 class FitWindow: public QWidget
8 {
9 protected:
10     virtual void paintEvent(QPaintEvent *);
11 };
12
13 #endif
```

FitWindow.

```
1 /* fitwindow.cpp */
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include "fitwindow.h"
5
6 int main(int argc,char **argv)
7 {
```

```
8      KApplication app(argc,argv,  "fitwindow");
9      FitWindow fitwindow;
10     fitwindow.show();
11     app.setMainWidget(&fitwindow);
12     return  (app.exec());
13 }
14 void FitWindow::paintEvent(QPaintEvent *)
15 {
16     QPainter p(this);
17
18     p.setWindow(0,0,300,300);
19
20     p.drawRoundRect(50,50,200,200,30,30);
21     p.setBrush(QColor("black"));
22     p.drawEllipse(100,100,100,100);
23     p.setBrush(QColor("white"));
24     p.drawPie(50,50,100,100,270*16,90*16);
25     p.drawPie(150,150,100,100,90*16,90*16);
26 }
```

18행의 setWindow()호출은 자리표계의 원점으로서 왼쪽웃구석을 설정한다. 또한 창문의 높이와 너비를 모두 300으로 설정한다. 창문에 그릴 때마다 300×300크기는 그림치수로 가정되지만 실제 화소들이 그려질 때 그것들은 실제창문크기로 변환된다. 그림 13-5는 2가지 다른 방법으로 크기를 조절한 후에 이 프로그람으로부터 현시된 창문을 보여준다.
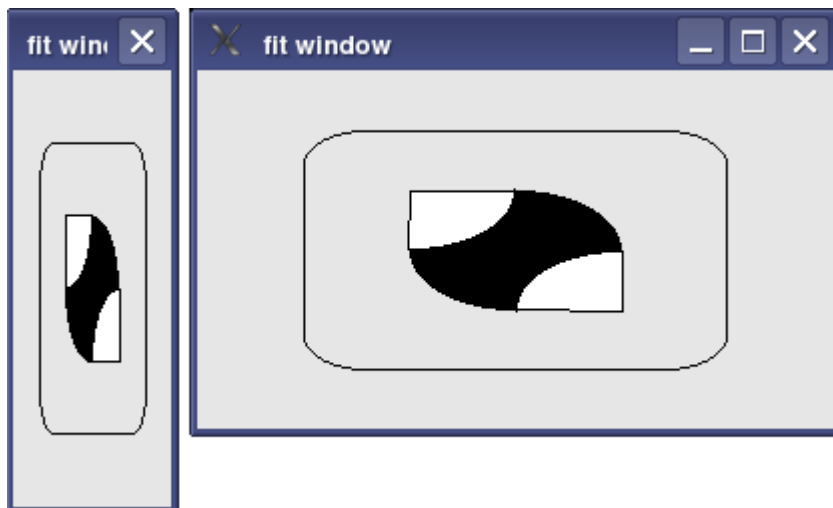
그림 13-5. 창문크기변화에 따르는 도형의 변화

제5절. 보조창문에 그림을 맞추기

이전 실례의 setWindow()메쏘드는 창문안의 보조창문에 맞게 그리기와 칠하기가 비례되도록 하기 위하여 setViewport()와 결합하여 사용할수 있다. 다음 실례는 창문안의 4개 보조창문으로 같은 그리기를 사영하여 그림 13-6에 보여주는 현시를 만든다.
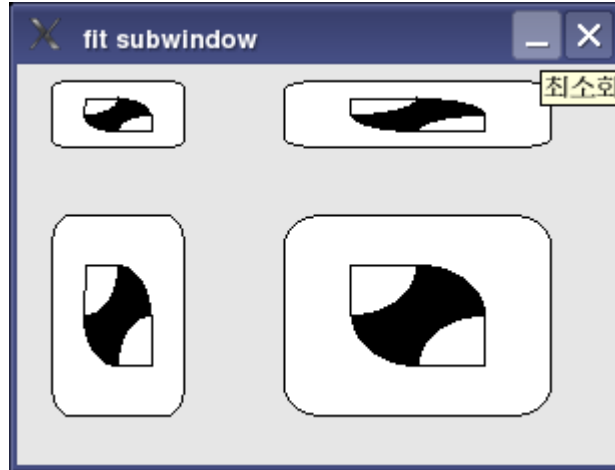
그림 13-6. 같은 창문에 4번 나타나는 같은 도형

FitSubWindow머리부파일

```
1 /* fitsubwindow.h */
2 #ifndef FITSUBWINDOW_H
3 #define FITSUBWINDOW_H
4
5 #include <qwidget.h>
6 #include <qpainter.h>
7
8 class FitSubWindow: public QWidget
9 {
10 public:
11     FitSubWindow(QWidget *parent=0,const char *name=0);
12 private:
13     void paintFigure(QPainter &);
14 protected:
15     virtual void paintEvent(QPaintEvent *);
16 };
17
18 #endif
```

FitSubWindow

```
1 /* fitsubwindow.cpp */
2 #include <kapplication.h>
3 #include "fitsubwindow.h"
4
5 int main(int argc,char **argv)
6 {
7       KApplication app(argc,argv,"fitsubwindow");
8       FitSubWindow fitsubwindow;
9       fitsubwindow.show();
10      app.setMainWidget(&fitsubwindow);
11      return  (app.exec());
12 }
13 FitSubWindow::FitSubWindow(QWidget *parent,
14      const char *name) : QWidget(parent,name)
15 {
16      setFixedSize(300,200);
17 }
18 void FitSubWindow::paintEvent(QPaintEvent *)
19 {
20      QPainter p(this);
21
22      p.setViewport(0,0,100,50);
23      paintFigure(p);
24      p.setViewport(100,0,200,50);
25      paintFigure(p);
26      p.setViewport(0,50,100,150);
27      paintFigure(p);
28      p.setViewport(100,50,200,150);
29      paintFigure(p);
30 }
31 void FitSubWindow::paintFigure(QPainter &p)
32 {
33      p.setWindow(0,0,300,300);
```

```
34    p.setBrush(QColor("white"));
35    p.drawRoundRect(50,50,200,200,30,30);
36    p.setBrush(QColor("black"));
37    p.drawEllipse(100,100,100,100);
38    p.setBrush(QColor("white"));
39    p.drawPie(50,50,100,100,270*16,90*16);
40    p.drawPie(150,150,100,100,90*16,90*16);
41 }
```

구성자는 16행의 setFixedSize()호출에 의해 300✕200화소 크기로 창문크기를 고정한다. 18행의 paintEvent()메쏘드는 창문이 그려질 때마다 호출되며 setViewport()와 paintFigure()의 4번의 호출이 그림 13-6에 보여준 창문을 그리게 한다.

22행의 setViewport()호출은 위치가 (0,0)인 왼쪽웃구석으로부터 도형을 그리며 100×50화소로 그리기령역을 제한한다. 23행의 paintFigure()호출은 화소들의 실제그리기를 진행한다. 같은 방식으로 setViewport()의 첫 호출에서 보조창문을 지정하고 그다음 도형을 그리는 paintFigure()를 호출함으로써 보기구역(보조창문)에 3가지 다른 도형을 그린다.

31행에서 메쏘드 paintFigure()은 도형을 그린다. 33행에서 setWindow()를 호출하여 창문이 왼쪽웃구석을 원점으로 하여 300×300화소크기의 바른4각형구역에 그려지도록 한다. 도형은 그다음 원시그리기메쏘드drawRoundRect(), drawEllipse(), drawPie()를 호출함으로써 300×300 정방형에 그려진다. 화소들의 물리적위치와 결과그림의 가로세로비는 paintFigure()호출에 앞서 setViewport()호출에 의해 설정된다. 이것은 paintFigure()메쏘드가 독립적으로 실제화소위치들의 도형을 계산하여 그리게 한다.

제6절. 오려내기

그리기를 특정한 령역으로 제한할수 있다. 즉 령역밖에서의 모든 그리기는 잘리우고 그려지지 않게 할수 있다. 다음 실례는 3개의 다른 솔을 사용하여 같은 타원을 그리지만 2번째와 3번째 타원은 원시타원이 여전히 부분적으로 보이도록 잘리운다.

ClipArea

```
1 /* cliparea.cpp */
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include <qbrush.h>
5 #include <qpointarray.h>
6 #include "cliparea.h"
7
8 int main(int argc,char **argv)
```

```
9 {
10    KApplication app(argc,argv,  "cliparea");
11    ClipArea cliparea;
12    cliparea.show();
13    app.setMainWidget(&cliparea);
14    return  (app.exec());
15 }
16 ClipArea::ClipArea(QWidget *parent,
17    const char *name) : QWidget(parent,name)
18 {
19    setFixedSize(300,200);
20 }
21 void ClipArea::paintEvent(QPaintEvent *)
22 {
23    QPainter p(this);
24
25    p.setBrush(QColor("white"));
26    p.drawEllipse(25,25,250,150);
27
28    p.setBrush(QBrush(QColor("black"),Qt::VerPattern));
29    p.setClipRect(30,30,70,70);
30    p.drawEllipse(25,25,250,150);
31
32    p.setBrush(QBrush(QColor("black"),Qt::Dense5Pattern));
33    QPointArray pa;
34    pa.setPoints(3,100,140,200,50,220,180);
35    QRegion region(pa);
36    p.setClipRegion(region);
37    p.drawEllipse(25,25,250,150);
38 }
```

25행의 setBrush()호출과 26행의 drawEllipse()호출은 그림 13-7과 같이 흑색륜곽으로 백색타원을 그린다.
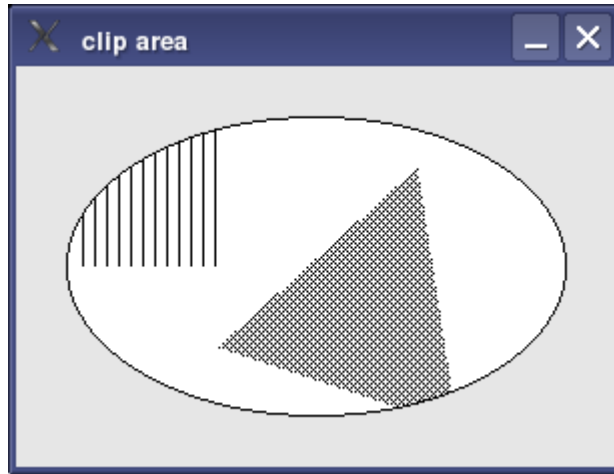
그림 13-7. 오려내기와 각이한 솔로 그린 타원

28～30행은 타원의 왼쪽웃구석에 수직선들을 그린다. 솔은 흑색VerPattern으로 설정되므로 수직선들이 만들어지며 29행의 setClipRect()호출은 70×70화소 크기의 직4각형구역으로 그리기를 제한하며 위치 (30,30)에 그 왼쪽구석이 있다. 30행의 drawEllipse()호출이 전체 도형을 그리지만 실제그리기는 오려낸 구역으로 제한된다. 또한 솔패턴이 배경을 보이게 하므로 원시타원의 부분도 역시 보인다.

같은 기술은 타원오른쪽의 3각형구역을 칠하는데 사용된다.

이런 경우에 잘라낸 령역은 36행의 setClipRegion()호출에 의해 정의된다. pa라는 QPointArray는 오직 3각형의 3개 점만 포함하지만 자기가 정의하려고하는 복잡한 다각형들도 포함할수 있다.

또한 1개의 오려낸 령역만 정의할수 있으므로 새로운 오려내기령역의 정의는 이전의 령역을 삭제한다. 오래낸 령역을 금지하려고 한다면 setClipping()메쏘드를 다음과 같이 호출할수 있다.

```
p.setClipping(FALSE);.
```

제7절. 확대축소

그림을 그리기에 앞서 QPainter에서 자리표들을 변경하여 크거나 작게 그림을 확대축소할수 있다. 다음 실례는 크기를 변경하고 픽스매프를 그린 결과를 보여준다:

ScaleShape머리부파일

```
1 /* scaleshape.h */
2 #ifndef SCALESHAPE_H
3 #define SCALESHAPE_H
4
5 #include <qwidget.h>
6
```

```
7 class ScaleShape: public QWidget
8 {
9 public:
10     ScaleShape(QWidget *parent=0,const char *name=0);
11 private:
12     QPixmap marble;
13 protected:
14     virtual void paintEvent(QPaintEvent *);
15 };
16
17 #endif
```

ScaleShape.

```
1 /* scaleshape.cpp */
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include "scaleshape.h"
5
6 #include "bluemarble.xpm"
7
8 int main(int argc,char **argv)
9 {
10     KApplication app(argc,argv,  "scaleshape");
11     ScaleShape scaleshape;
12     scaleshape.show();
13     app.setMainWidget(&scaleshape);
14     return  (app.exec());
15 }
16 ScaleShape::ScaleShape(QWidget *parent,const
17     char *name) : QWidget(parent,name)
18 {
19     marble = QPixmap(magick);
20     setFixedSize(360,288);
21 }
22 void ScaleShape::paintEvent(QPaintEvent *)
23 {
```

```
24    QPainter p(this);
25
26    p.drawPixmap(0,0,marble);
27    p.scale(2.0,2.0);
28    p.drawPixmap(36,0,marble);
29    p.scale(1.0,2.0);
30    p.drawPixmap(108,0,marble);
31 }
```

6행에 포함된 XPM파일은 19행에서 QPixmap로 변환된다. 픽스매프는 72×72화소이다.

26행의 drawPixmap()호출은 그림 13-8과 같이 창문의 왼쪽웃구석에 본래 크기로 픽스매프를 그린다.

기정비례값들은 x와 y축에서 모두 1.0이다. scale()은 두 값의 2배크기로 호출한다. 27행의 scale()호출은 x와 y방향의 비례가 2배하므로 28행의 drawPixmap()호출은 픽스매프를 처음것의 2배로 그린다. drawPixmap()호출에서 본래크기의 2배 즉 72화소로 확대된 그림을 자리표의 오른쪽으로 36화소만큼 옮긴다. 29행의 scale()호출은 현재 비례에 2.0이 아니라 1.0을 곱하므로 x축의 비례를 변경하지 않지만 y축의 비례를 두배로 하여 수직비례요소가 4.0이 되게 한다. 30행의 drawPixmap()호출은 픽스매프를 원래 크기의 두배의 너비와 4배의 높이로 그린다.
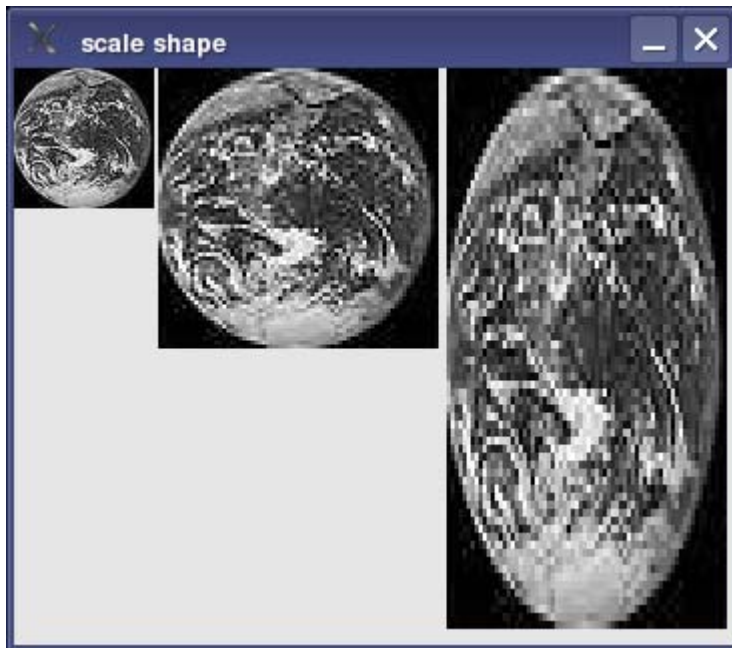
그림 13-8. 3개의 각이한 비례로 그린 픽스매프

제8절. 경사지기

도형을 경사지게 한다는것은 x축과 수평으로 놓이지 않거나 y축과 수직으로 놓이지 않거나 또는 x와 y축에 모두 수평 및 수직으로 놓이지 않도록 도형이 기울어지게 하는것이다. y축의 경사도를 증가시키면 도형의 아래를 오른쪽으로 이동하고 x축의 경사도를 증가시키면 도형의 오른쪽끝을 아래로 이동한다. 이동량은 창문크기에 따라 결정된다. 부의 경사도는 축을 반대방향으로 이동한다. 실례로 창문이 100화소너비이면 x축의 경사도 0.5는 그림의 오른변을 50화소 아래로 이동하며 1.0의 경사도는 오른변을 100화소 아래로 이동하며 경사도 -1.0은 오른쪽을 100화소 우로 이동한다.

다음 실례는 x와 y방향 모두에서 경사지게 한 결과를 보여준다. 그림 13-9에서는 x경사도 1.0, 경사도없음, 그리고 y경사도 1.0으로 그린 같은 도형들(왼쪽에서 오른쪽으로)을 보여준다.
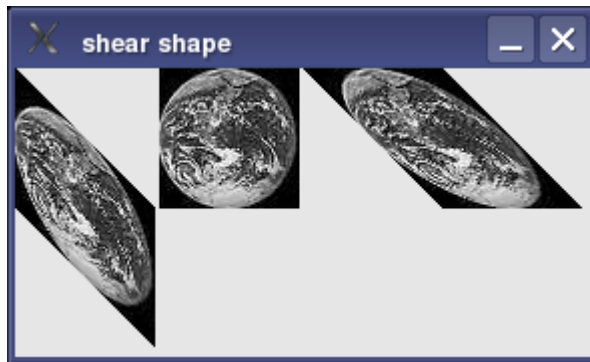
그림 13-9. 수직경사지기, 경사없음, 수평경사지기

ShearShape머리부파일

```
1 /* shearshape.h */
2 #ifndef SHEARSHAPE_H
3 #define SHEARSHAPE_H
4
5 #include <qwidget.h>
6
7 class ShearShape: public QWidget
8 {
9 public:
10     ShearShape(QWidget *parent=0,const char *name=0);
11 private:
12     QPixmap marble;
13 protected:
```

```
14    virtual void paintEvent(QPaintEvent *);
15 };
16
17 #endif
```

ShearShape

```
1 /* shearshape.cpp */
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include "shearshape.h"
5
6 #include "bluemarble.xpm"
7
8 int main(int argc,char **argv)
9 {
10    KApplication app(argc,argv,  "shearshape");
11    ShearShape shearshape;
12    shearshape.show();
13    app.setMainWidget(&shearshape);
14    return  (app.exec());
15 }
16 ShearShape::ShearShape(QWidget *parent,const
17    char *name) : QWidget(parent,name)
18 {
19    marble = QPixmap(magick);
20    setFixedSize(288,144);
21 }
22 void ShearShape::paintEvent(QPaintEvent *)
23 {
24    QPainter p(this);
25
26    p.shear(1.0,0.0);
27    p.drawPixmap(0,0,marble);
28    p.shear(-1.0,0.0);
29    p.drawPixmap(72,0,marble);
30    p.shear(0.0,1.0);
```

```
31    p.drawPixmap(144,0,marble);
32 }
```

6행에 포함된 XPM파일은 19행에서 QPixmap로 변환된다. 26행의 shear()호출은 모든 수평선들이 오른쪽으로 45도 이울어져서 그려지도록 x축의 오른쪽끝을 낮춘다. 27행의 drawPixmap()호출결과는 그림 13-9의 왼쪽에 있는 경사진 픽스매프이다.

28행의 shear()호출은 이전의 shear()호출의 동작을 반대로 하며 29행의 drawPixmap()호출에 의해 도형은 경사지지 않고 그려진다. 30행의 shear()호출은 y축의 경사도를 조절하고 31행의 drawPixmap()호출은 그림 13-9의 오른쪽에 보여주는 픽스매프를 생성한다.

제9절. 변환

그리기의 원점은 보통 창문의 왼쪽웃구석이지만 다른 장소로 옮길수 있다. 일단 옮기면 y값은 원점아래로 옮겨지고 x값은 오른쪽으로 옮겨지며 원점의 우와 왼쪽의 위치들은 부의 자리표값으로 설정된다. 다음 실례는 원점을 창문중심으로 변환하고 그 주위에 픽스매프들을 현시한다.

TranslateShape머리부파일

```
1 /* translateshape.h */
2 #ifndef TRANSLATESHAPE_H
3 #define TRANSLATESHAPE_H
4
5 #include <qwidget.h>
6
7 class TranslateShape: public QWidget
8 {
9 public:
10    TranslateShape(QWidget *parent=0,const char *name=0);
11 private:
12    QPixmap marble;
13 protected:
14    virtual void paintEvent(QPaintEvent *);
15 };
16
17 #endif
```

TranslateShape

```
1 /* translateshape.cpp */
2 #include <kapplication.h>
```

```
3 #include <qpainter.h>
4 #include "translateshape.h"
5
6 #include "bluemarble.xpm"
7
8 int main(int argc,char **argv)
9 {
10    KApplication app(argc,argv,  "translateshape");
11    TranslateShape translateshape;
12    translateshape.show();
13    app.setMainWidget(&translateshape);
14    return  (app.exec());
15 }
16 TranslateShape::TranslateShape(QWidget *parent,const
17    char *name) : QWidget(parent,name)
18 {
19    marble = QPixmap(magick);
20    setFixedSize(180,180);
21 }
22 void TranslateShape::paintEvent(QPaintEvent *)
23 {
24    QPainter p(this);
25
26    p.translate(90,90);
27    p.drawPixmap(-72,-72,marble);
28    p.drawPixmap(0,0,marble);
29 }
```

6행에 포함된 XPM파일은 19행에서 QPixmap로 변환된다. 26행의 translate()호출은 창문의 위치 (90,90)으로 모든 그림의 원점을 이동한다. 변환된 원점을 리용하여 27행의 drawPixmap호출은 창문에서 보통 위치 (18,18)로 픽스매프의 왼쪽웃구석을 배치한다. 28행의 drawPixmap()호출은 픽스매프의 왼쪽웃구석을 새로운 원점 즉 (90,90)인 점에 배치한다. 결과 창문은 그림 13-10에서 보여준다.

그림 13-10. 새 원점으로부터 픽스매프들의 배치

제10절. 회전

전체자리표계를 원점주위로 회전할수 있다. 표시장치에서 기정원점은 왼쪽웃구석으로서 일반적인 도형현시에서 회전중심의 유효위치로 리용하지 않는다. 다음 실례는 원점변환을 리용하여 픽스매프의 중심에 회전중심을 배치한다.

RotateShape머리부파일

```
1 /* rotateshape.h */
2 #ifndef ROTATESHAPE_H
3 #define ROTATESHAPE_H
4
5 #include <qwidget.h>
6
7 class RotateShape: public QWidget
8 {
9 public:
10     RotateShape(QWidget *parent=0,const char *name=0);
11 private:
12     QPixmap marble;
13 protected:
14     virtual void paintEvent(QPaintEvent *);
15 };
16
17 #endif
```

RotateShape

```
1 /* rotateshape.cpp */
```

```
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include "rotateshape.h"
5
6 #include "bluemarble.xpm"
7
8 int main(int argc,char **argv)
9 {
10     KApplication app(argc,argv,  "rotateshape");
11     RotateShape rotateshape;
12     rotateshape.show();
13     app.setMainWidget(&rotateshape);
14     return  (app.exec());
15 }
16 RotateShape::RotateShape(QWidget *parent,const
17     char *name) : QWidget(parent,name)
18 {
19     marble = QPixmap(magick);
20     setFixedSize(320,160);
21 }
22 void RotateShape::paintEvent(QPaintEvent *)
23 {
24     QPainter p(this);
25
26     p.translate(80,80);
27     p.rotate(20.0);
28     p.drawPixmap(-36,-36,marble);
29     p.rotate(-20.0);
30     p.translate(80,0);
31     p.rotate(40.0);
32     p.drawPixmap(-36,-36,marble);
33     p.rotate(-40.0);
34     p.translate(80,0);
35     p.rotate(60.0);
36     p.drawPixmap(-36,-36,marble);
```

37 }

6행의 XPM파일은 19행에서 QPixmap로 변환된다. 26행의 translate()호출은 창문의 위치 (80, 80)에로 원점을 이동한다. 27행의 rotate()호출은 원점주위의 자리표계를 20° 회전시킨다. 28행의 drawPixmap()호출은 원점을 중심으로 하여 픽스매프를 그린다(픽스매프는 72×72화소크기).

그림 13-11에 보여준것처럼 두번째 픽스매프그림은 첫째의 오른쪽에 중심에 직접 배치된다. 이전 회전설정이 남아있으면 30행에서 원점을 20°각도로 오른쪽과 아래로 변화시킨다. 이것을 막기 위하여 회전값을 반대로 0으로 설정함으로써 이전의 회전을 반전시키고 30행에서 translate()호출로 원점을 오른쪽으로 직접 이동하게 한다. 원점이 이동한 후에 31행의 rotate()호출은 둘째 픽스매프의 회전량을 40°로 설정하고 32행의 drawPixmap()호출에 의해 그려진다. 같은 방법으로 회전은 33행에서 삭제하여 34행에서 translate ()호출을 허용하고 원점을 다시 한번 오른쪽으로 이동한다. 그다음 회전은 마지막 drawPixmap()호출을 위해 35행에서 60°로 설정된다.
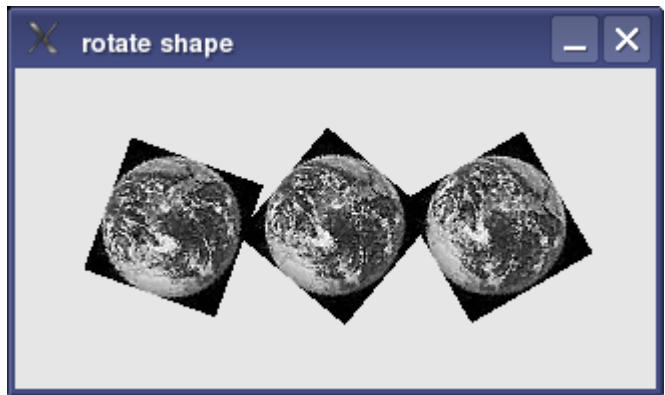
그림 13-11. 3개 원점주위로 픽스매프들을 회전시키기

제11절. 2차베쎌곡선

QPointArray클라스는 4개의 점으로 베쎌곡선을 생성하는데 쓰이는 메쏘드를 가지고있다. 그림 13-12는 아래의 프로그람의 실행결과를 보여준다.
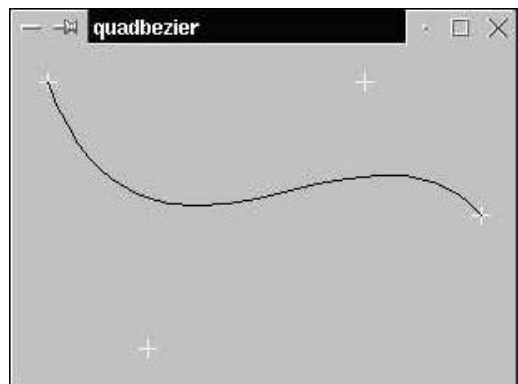
그림 13-12. 4개의 점으로 만들어진 베쎌곡선

QuadBezier

```
1 /* quadbezier.cpp */
2 #include <kapplication.h>
3 #include <qpen.h>
4 #include "quadbezier.h"
5
6 int main(int argc,char **argv)
7 {
8     KApplication app(argc,argv,"quadbezier");
9     QuadBezier quadbezier;
10    quadbezier.show();
11    app.setMainWidget(&quadbezier);
12    return  (app.exec());
13 }
14 QuadBezier::QuadBezier(QWidget *parent,
15    const char *name) : QWidget(parent,name)
16 {
17    setFixedSize(300,200);
18 }
19 void QuadBezier::paintEvent(QPaintEvent *)
20 {
21    static QCOORD points[] =
22             { 20,20, 80,180, 210,20, 280,100 };
23    QPointArray pa(4,points);
24    QPainter p(this);
25
26    p.setPen(QColor("white"));
27    paintPoints(p,pa);
28    QPointArray bpa = pa.quadBezier();
29    p.setPen(QColor("black"));
30    p.drawPolyline(bpa);
31 }
32 void QuadBezier::paintPoints(QPainter &p,QPointArray &pa)
33 {
34    int x;
```

```
35    int y;
36
37    for(int i=0; i<pa.size(); i++) {
38            pa.point(i,&x,&y);
39            p.drawLine(x-5,y,x+5,y);
40            p.drawLine(x,y-5,x,y+5);
41    }
42 }
```

23행에서 pa라는 QPointArray객체는 4개의 점을 포함하도록 창조된다. 26행은 백색펜을 지정하므로 27행의 paintPoints()호출은 백색으로 점위치들을 지적한다. 28행의 quadBezier()호출은 4개의 원점을 사용하여 점모임을 포함하는 새로운 QPointArray객체를 만들며 첫점과 넷째 점이 그 점들사이의 2차베쎌곡선의 궤적을 표현하고 2개 중점에 의해 모양이 정해진다. 29행과 30행에서 setPen()과 drawPolyline()호출은 곡선을 흑색으로 그린다.

32행의 paintPoints()메쏘드는 QPointArray에서 매 점의 위치에 10×10화소크기의 《+》문자를 배치한다. 순환에서 point()호출에 의해 점의 x와 y값들이 되돌아오고 그다음 drawLine()를 두번 호출하는데 한번은 수직선을 그리고 다음에는 수평선을 그린다. 이리하여 점을 표시한다.

제12절. 픽스매프렬에 의한 동화

동화처리는 프레임들을 하나씩 현시하는 문제이다. 이 프레임들은 파일로부터 적재된 화상들의 렬(영화부류)일수 있으며 또는 표준도형함수를 사용하여 그릴수 있다. 다음 실례는 픽스매프를 사용하여 그림 13-13에 보여준 튀는 공들을 그리고 동화시킨다.
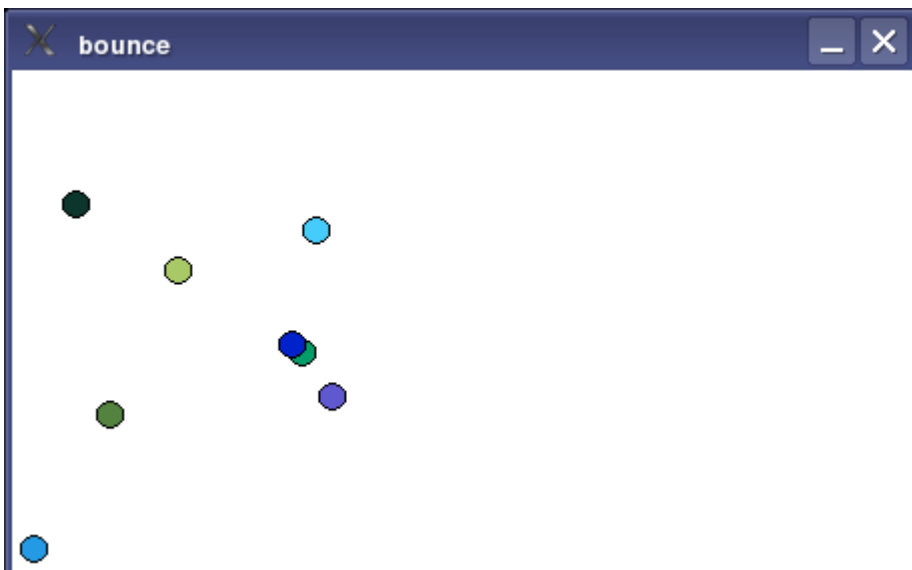
그림 13-13. 튀는공의 동화

Ball머리부파일

```
1 /* ball.h */
2 #ifndef BALL_H
3 #define BALL_H
4
5 #include <qcolor.h>
6
7 #define RADIUS 7
8
9 class Ball
10 {
11 public:
12     Ball(int width,int height);
13 private:
14     double x;
15     double y;
16     double xVelocity;
17     double yVelocity;
18     QColor *color;
19 public:
20     double getDiameter() { return(RADIUS * 2.0); }
21     double getX() { return(x); }
22     double getY() { return(y); }
23     double getXVelocity() { return(xVelocity); }
24     double getYVelocity() { return(yVelocity); }
25     QColor &getColor() { return(*color); }
26     void setXVelocity(double value) { xVelocity = value; }
27     void setYVelocity(double value) { yVelocity = value; }
28     void nextPosition();
29 };
30
31 #endif
```

매개 공은 7행에서 정의한것처럼 같은 반경을 가진다. 어떤 주어진 순간에 창문에서 공의 위치는 14행과 15행에서 값 x와 y에 의해 지정된다. 공의 수평속도는 xVelocity에 의해 지정되는데 이 값은 오른쪽으로 이동하면 정수로, 왼쪽으로 이동하면 부수로 된다. 공이 아

래로 움직이고 있으면 yVelocity의 수직속도값은 정수이고 우로 움직이고 있으면 부수이다. 매개 공은 18행에서 정의된것처럼 자체의 색을 가지고있다.

20～25행에서 정의된 메쏘드들은 Ball클라스에 보관된 값의 호출을 제공한다. 26행과 27행의 메쏘드 setXVelocity()와 setYVelocity()는 공의 속도를 설정하는데 쓰인다.

Ball

```
1 /* ball.cpp */
2 #include <stdlib.h>
3 #include "ball.h"
4
5 Ball::Ball(int width,int height)
6 {
7       x = (((double)rand()*width)/RAND_MAX);
8       y = (((double)rand()*height)/RAND_MAX) - height;
9       yVelocity = 0.0;
10      do {
11              xVelocity = (((double)rand()*4)/RAND_MAX) - 2;
12      } while(fabs(xVelocity) < 0.5);
13      color = new QColor(rand() % 255,
14      rand() % 255,
15      rand() % 255);
16 }
17 void Ball::nextPosition()
18 {
19      x += xVelocity;
20      y += yVelocity;
21 }
```

5행의 구성자는 위치, 속도, 공의 색을 우연설정값으로 초기화한다. 공은 창문의 두 경계사이에서 수평으로 그리고 창문의 어떤 위치에서 수직으로 배치되지만 보임창문높이의 2배이상으로 되지 않는다. 수직속도는 중력이 공을 아래로 가속시키므로 0으로 설정된다. 10행의 순환은 수평속도가 공이 수직으로 너무 길게 튀지 않도록 일정하게 한다. 색은 우연 RGB값으로 설정된다.

17행의 nextPosition()메쏘드는 공을 현재상태에서 다음 위치에로 이동하기 위해 호출된다. 이것은 간단히 속도량을 현재위치에 추가하여 수행한다. 속도의 크기는 공이 얼마나 멀리 움직이는가를 결정하며 속도의 부호는 그 방향을 결정한다.

Bounce머리부파일

```
1 /* bounce.h */
2 #ifndef BOUNCE_H
3 #define BOUNCE_H
4
5 #include <qwidget.h>
6 #include <qtimer.h>
7 #include <qpixmap.h>
8 #include "ball.h"
9
10 #define BOUNCE_HEIGHT 250
11 #define BOUNCE_WIDTH 450
12 #define BALL_COUNT 10
13 #define GRAVITY 0.2
14
15 class Bounce: public QWidget
16 {
17     Q_OBJECT
18 public:
19     Bounce(QWidget *parent=0,const char *name=0);
20 private:
21     QTimer *timer;
22     QPixmap *pixmap;
23     Ball *ball[BALL_COUNT];
24 private slots:
25     void frameSlot();
26 protected:
27     virtual void paintEvent(QPaintEvent *);
28 };
29
30 #endif
```

Bounce클라스는 공들을 현시하는데 사용될 창문을 포함하는 창문부품이다. 10∼13행에서 정의된 상수값들은 창문의 높이와 너비, 공의 개수, 그리고 공을 아래로 가속시키는 중력을 결정한다.

21행에서 정의한 QTimer는 프레임들사이의 시격을 지정하는데 사용된다. 22행의 QPixmap는 창문에 매개 프레임을 현시하기전에 그것을 그리는데 사용된다. 23행은 지적자배

렬을 선언한다.

Bounce

```
1 /* bounce.cpp */
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include "bounce.h"
5
6 int main(int argc,char **argv)
7 {
8     KApplication app(argc,argv, "bounce");
9     Bounce bounce;
10    bounce.show();
11    app.setMainWidget(&bounce);
12    return (app.exec());
13 }
14 Bounce::Bounce(QWidget *parent,const
15    char *name) : QWidget(parent,name)
16 {
17    setFixedSize(BOUNCE_WIDTH,BOUNCE_HEIGHT);
18    timer = new QTimer(this, "clock");
19    connect(timer,SIGNAL(timeout()),
20          this,SLOT(frameSlot()));
21    timer->start(20);
22    for(int i=0; i<BALL_COUNT; i++)
23          ball[i] = new Ball(BOUNCE_WIDTH,BOUNCE_HEIGHT);
24    pixmap = new QPixmap(BOUNCE_WIDTH,BOUNCE_HEIGHT);
25 }
26 void Bounce::paintEvent(QPaintEvent *)
27 {
28    QPainter paint;
29    Ball *b;
30
31    paint.begin(pixmap);
32    paint.eraseRect(0,0,BOUNCE_WIDTH,BOUNCE_HEIGHT);
33    for(int i=0; i<BALL_COUNT; i++) {
```

```
34          b = ball[i];
35          paint.setBrush(b->getColor());
36          paint.drawEllipse(b->getX(),b->getY(),
37          b->getDiameter(),b->getDiameter());
38     }
39     paint.end();
40     bitBlt(this,0,0,pixmap,0,0,
41          BOUNCE_WIDTH,BOUNCE_HEIGHT,CopyROP);
42 }
43 void Bounce::frameSlot()
44 {
45     Ball *b;
46
47     for(int i=0; i<BALL_COUNT; i++) {
48          b = ball[i];
49          if((b->getX() >= BOUNCE_WIDTH) ||
50                    (b->getX() < -b->getDiameter())) {
51               delete b;
52               ball[i] = new Ball(BOUNCE_WIDTH,BOUNCE_HEIGHT);
53               continue;
54          }
55          if(b->getY() + b->getDiameter() >= BOUNCE_HEIGHT) {
56               if(b->getYVelocity() > 0)
57                    b->setYVelocity(-b->getYVelocity() * 0.9);
58          } else {
59               b->setYVelocity(b->getYVelocity() + GRAVITY);
60          }
61          b->nextPosition();
62     }
63     repaint(0,0,BOUNCE_WIDTH,BOUNCE_HEIGHT,FALSE);
64 }
```

14행에서 시작하는 Bounce구성자는 동화에 사용될 모든 객체를 만든다. QTimer는 동화의 속도와 프레임들사이에 경과시간의 비률을 조종하며 18행에서 창조된다. 19행의 connect()호출은 처리부메쏘드 frameSlot()에 시계를 련결하여 시계가 끝날 때마다 처리부가 호출되게 한다. 20행의 start()호출은 처리부메쏘드를 20ms(0.20s)마다 호출하도록 한다. 22행의 순환은

동화에서 사용할 공들을 만들고 작업픽스매프는 24행에서 창조된다.

paintEvent()호출은 매개 공을 창문에서 그 현재 위치에 그린다. 그리기는 창문부품의 창문이 아니라 픽스매프에 수행된다. 32행의 eraseRect()호출은 픽스매프를 지운다. 매개 공의 상세한 그리기는 33행에서 시작하는 순환에서 수행된다. 35행의 setBrush()호출은 현재솔을 공의 색으로 설정하고 drawEllipse()호출은 공객체의 정보를 리용하여 적합한 위치에서 색칠된 원을 그린다.

40행에서 bitBlt()함수호출은 창문부품의 창문에 직접 전체 픽스매프를 복사한다. 그리기를 창문부품창문우에서 직접 진행하고 창문의 지우기와 다시그리기는 창문이 깜박거리게 한다. 픽스매프그리기와 블로크로서의 화소복사는 깜빡거림을 제거한다. bitBlt()에 넘긴 처음 3개 인수는 원천을, 다음 3개는 목적지, 그리고 다음 2개는 복사될 직4각형의 크기를 지정한다. 마지막 인수는 화소자료의 전송방법을 지정한다. CopyROP선택은 매개 목적화소가 간단히 대응하는 원천화소에 다시 써진다는것을 지정한다. 기타 많은 선택이 있는데 표 13-1에 보여준다.

표 13-1. bitBlt()에 의한 화소복사에 사용할수 있는 라스터조작

이름	목적지
AndNotROP	Source AND (NOT Destination)
AndROP	Source AND Destination
ClearROP	0
CopyROP	Source
NandROP	NOT (Source AND Destination)
NopROP	Destination
NorROP	NOT (Source OR Destination)
NotAndROP	(NOT Source) AND Destination
NotCopyROP	NOT Source
NotOrROP	(NOT Source) AND Destination
NotROP	NOT Destination
NotXorROP	(NOT Source) XOR Destination
OrNotROP	Source OR (NOT Destination)
OrROP	Source OR Destination
SetROP	1
XorROP	Source XOR Destination

처리부메쏘드 frameSlot()는 시계가 끝날 때마다 호출된다. 이 메쏘드는 실제로 새 창문을 그리지 않지만 매개 공들의 다음 위치를 계산하고 그다음 63행에서 repaint()를 호출하여 paintEvent()호출을 예정한다. repaint()호출은 그리려는 직4각형을 지정하고 (이 정보는 QPaintEvent인수내에서 paintEvent()에로 넘어간다) 또한 창문이 처음에 지워지는가 하는것을

지정한다. 동화에서 창문지우기선택은 그렇게 하는것이 깜박거림을 다시 발생시키므로 FALSE로 설정된다.

47행에서 시작하는 순환은 매개 공의 다음 위치를 결정한다. 49행의 조건이 TRUE이면 공은 오른쪽 또는 왼쪽으로 창문을 벗어나므로 그것은 삭제되고 새로운 공으로 교체된다. 55행은 공이 창문의 바닥에 있는가 하는것과 아래로 이동하고있는가(즉 그 수직속도가 정수이다) 결정하며 만일 그렇다면 속도를 반전하여 공이 우로 올라가게 한다. 속도는 10%만큼 감소되므로(쓸림으로 인한 에네르기손실) 공은 매번 전보다 낮게 튀여오른다. 공이 창문에서 비행하고 있으면 59행의 식은 중력의 량만큼 속도를 떨군다. 끝으로 61행의 nextPosition()호출은 그 현재속도에 따라서 공의 x와 y위치를 조정한다.

제13절. QImage에 의한 화소값호출

QImage객체는 화상정보를 보관하고 개별적인 화소정보에 대한 저준위호출을 제공하는데 사용된다. 다음 실례는 QPixmap를 창조하고 그것을 QImage로 변환하여 화소색값을 수정하며 그것을 다시 현시용의 QPixmap로 역변환한다. QImage의 3가지 형식이 있다. 즉 화소당 1bit, 8bit, 또는 32bit를 포함할수 있다.

QImage객체가 화소당 1bit만 포함한다면 그다음 QImage는 오직 흑백색도형정보를 가진다. 실제로 1bit값은 보통 흑백색을 포함하는 색략도의 색인으로 사용되지만 그것은 임의의 2가지 색을 포함할수 있다. 이런 형의 QImage객체에 색메쏘드와 기발들을 적용할수 있지만 그것들은 효과가 없다.

사용되는 색모형에 따라 QImage객체는 색략도에, 또는 매개 화소위치에 직접 실제색자료를 기억할수 있다(11장에 설명). 프로그람은 어느 한 경우에 색을 변경할수 있다. 색략도를 사용한다면 색략도의 색인 또는 색략도자체의 내용을 변경할수 있다. 색략도가 사용되지 않는다면 매개의 개별적인 화소를 변경할수도 있다.

ImageModify머리부파일

```
1 /* imagemodify.h */
2 #ifndef IMAGEMODIFY_H
3 #define IMAGEMODIFY_H
4
5 #include <qwidget.h>
6
7 class ImageModify: public QWidget
8 {
9 public:
10    ImageModify(QWidget *parent=0,const char *name=0);
11 private:
```

```
12    QPixmap logo;
13    QPixmap modlogo;
14    QRgb rgbModify(QRgb rgb);
15 protected:
16    virtual void paintEvent(QPaintEvent *);
17 };
18
19 #endif
```

12행의 logo픽스매프는 원시픽스매프를 보관하는데 사용되며 13행의 modlogo는 변경된 픽스매프를 보관하는데 사용된다.

ImageModify

```
1 /* imagemodify.cpp */
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include <qimage.h>
5 #include <qcolor.h>
6 #include "imagemodify.h"
7
8 #include "logo.xpm"
9
10 int main(int argc,char **argv)
11 {
12    KApplication app(argc,argv,  "imagemodify");
13    ImageModify imagemodify;
14    imagemodify.show();
15    app.setMainWidget(&imagemodify);
16    return(app.exec());
17 }
18 ImageModify::ImageModify(QWidget *parent,const
19    char *name) : QWidget(parent,name)
20 {
21    logo = QPixmap(magick);
22    QImage image = logo.convertToImage();
23    if(image.numColors() > 0) {
24          for(int i=0; i<image.numColors(); i++) {
```

```
25                  QRgb rgbOrig = image.color(i);
26                  QRgb rgbMod = rgbModify(rgbOrig);
27                  image.setColor(i,rgbMod);
28            }
29      } else {
30            for(int x=0; x<image.width(); x++) {
31                  for(int y=0; y<image.height(); y++) {
32                        QRgb rgbOrig = image.pixel(x,y);
33                        QRgb rgbMod = rgbModify(rgbOrig);
34                        image.setPixel(x,y,rgbMod);
35                  }
36            }
37      }
38      modlogo.convertFromImage(image,ThresholdDither);
39      setFixedSize(514,303);
40 }
41 QRgb ImageModify::rgbModify(QRgb rgb) {
42      int alpha = rgb & 0xFF000000;
43      QRgb rgbMod = qRgb(qGreen(rgb) & 0xC0,
44      qRed(rgb) & 0xC0,
45      qBlue(rgb) & 0xC0);
46      rgbMod |= alpha;
47      return  (rgbMod);
48 }
49 void ImageModify::paintEvent(QPaintEvent *)
50 {
51      QPainter p(this);
52
53      p.drawPixmap(0,0,logo);
54      p.drawPixmap(257,0,modlogo);
55 }
```

18행에서 시작하는 구성자는 2개 픽스매프를 만든다. logo라는 픽스매프는 8행에서 포함된 XPM자료로부터 21행에서 창조된다. 22행의 convertToImage()호출은 QPixmap객체의 내용을 리용하여 QImage객체를 만든다.

23행은 색모형을 결정한다. numColors()로부터 돌아온 값은 색략도에 보관된 색의 값이

다. 수가 0이면 색략도는 없고 화소색은 직접 변경하여야 한다.

24행에서 tj는 순환이 색략도의 매개 항목에 대하여 한번만 실행되게 한다. 25행의 메쏘드 color()는 QRgb값으로서 색화소값을 얻는다. QRgb자료형은 화소정보를 포함하는 unsigned 옹근수이다(11장에서 설명). 즉 제일 왼쪽 바이트는 알파(투명성)값을 보관하고 다른 3byte는 각각 3가지 색값을 하나씩 보관한다. 26행의 rgbModify()호출은 화상으로부터 QRgb값을 사용하여 새로운 QRgb값을 창조하여 돌려준다. 27행의 setColor()호출은 색표에서 같은 색인위치에 변경된 색값들을 보관한다. 색략도에 보관된 색값들을 직접 호출하므로 프로그람은 자기가 요구하는 변경을 할수 있다.

색략도가 없다면 색은 QImage의 매개 화소에 직접 보관된다. 30행과 31행의 순환은 모든 화소들을 순환하면서 QImage의 높이와 너비값을 사용한다. 실제변환은 rgbModify()메쏘드에 의하여 수행된다. 같은 메쏘드는 두 색모형의 인수들을 바꾸는데 사용된다. 색값들은 32행에서 pixel()호출에 의해 QImage객체로부터 읽어들이며 변경된 QRgb값들은 34행의 setPixel()호출에 의해 QImage객체에 써넣기된다.

38행의 convertFromImage()호출은 수정된 화상자료를 픽스매프로 역변환하여 후에 현시할수 있게 한다. 메쏘드의 첫 인수는 QImage객체이고 둘째 인수는 변환과정을 조종하는 기발들의 모임이다. 기발들은 표 13-2, 13-3, 13-4, 그리고 13-5에서 보여주었으며 이 기발들 결합하여 사용할수 있다.

표 13-2. QPixmap를 창조하기 위한 색선택기발

기발이름	설명
AutoColor	이것은 기정이다. 화소당 1bit이면 결과QPixmap는 흑백색이고 그렇지 않으면 한정값표현이 원색깊이로 진행되고 변환된다.
ColorOnly	QPixmap는 한정값표현이 원색깊이로 진행되고 변환된다.
MonoOnly	결과QPixmap는 흑백색이다.

표 13-3. QPixmap를 창조하기 위한 한정값표현법선택기발

기발이름	설명
DiffuseDither	이것은 기정값이다. 이것은 고품질결과를 산출하도록 설계된 한정값표현알고리듬이다.
OrderedDither	이것은 속도와 효률을 위해 설계한 한정값표현알고리듬이다.
ThresholdDither	이 알고리듬은 한정값표현은 진행되지 않는다. 가장 가까운 색이 사용된다.

표 13-4. QPixmap를 만들기 위한 알파통로 한정값표현방식선택기발

기발이름	설명
DiffuseAlphaDither	이것은 고품질결과를 산출하도록 설계된 한정값표현알고리듬이

	다.
OrderedAlphaDither	이것은 속도와 효률을 위해 설계된 한정값표현알고리듬이다.
ThresholdAlphaDither	이것은 기정으로서 한정값표현은 진행되지 않는다.

표 13-5. QPixmap를 만들기 위한 색생성선택기발

기발이름	설명
PreferDither	이것은 기정값이다. 32bit화상들이 8bit화상으로 변환되고있을 때 항상 32bit화상들을 한정값표현한다.
AvoidDither	8bit화상들로 변환되고있는 256개이상의 색을 포함하는 32bit 화상들을 한정값표현한다.

41행의 메쏘드 rgbModify()는 실제색변환을 진행한다. 알파값은 색값의 첫바이트에 보관되고 이 변환방법은 가변알파에 값을 보관하므로 후에 값을 변경된 색으로 되살릴수 있다. 현존투명성정보를 얻으려고 할 때만 이것이 필요하다. 3개 표식 qGreen, qRed 그리고 qBlue는 3가지 색값을 각각 얻기 위한것이다. 매개 색은 변경되고(값 0xC0과 비트별AND합으로써) 표식 qRgb()는 3개의 개별적인 색값들을 rgbMod라는 단일한 QRgb자료형으로 결합한다. 원시알파값이 추가된다. 이 실례는 매개 색값으로부터 처음 2bit를 간단히 삭제하여 전체색분해능을 6bit(매개 색에 둘)로 효과적으로 줄인다. 또한 적색과 록색의 값이 바꿔여진다. 그림 13-14에서는 왼쪽에 원시픽스매프, 오른쪽에서 변경된 픽스매프를 현시한다.
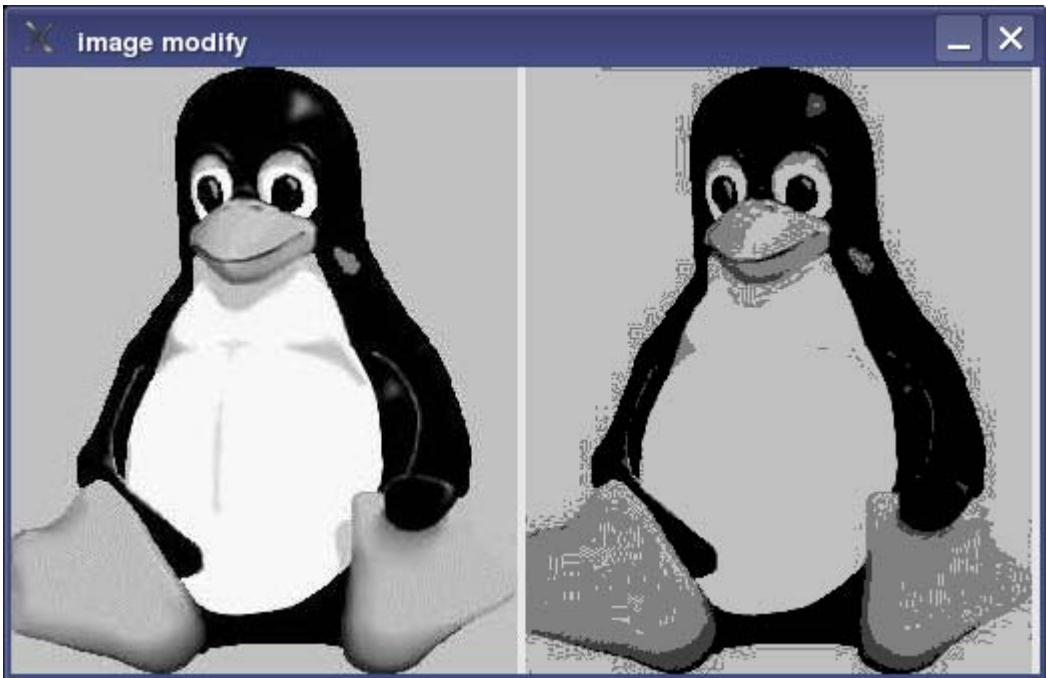
그림 13-14. 알파를 적용한 실례

화상에 적용될수 있는 색변환에는 제한이 없다. 실례로 모든 색정보를 삭제하고 회색의

도형을 만드는것은 간단히 적, 록, 청값들을 평균함으로써 회색의 그늘을 만드는 문제이다. 다음 rgbModify()메쏘드는 알파투명도를 얻고 회색의 화상을 생성한다.

```
QRgb ImageModify2::rgbModify(QRgb rgb) {
    int alpha = rgb & 0xFF000000;
    int average = qGreen(rgb) + qRed(rgb) + qBlue(rgb);
    average /= 3;
    QRgb rgbMod = qRgb(average,average,average);
    rgbMod |= alpha;
    return(rgbMod);
}
```

제14절. QFileDialog에서 그림기호제공기의 리용

매개 파일형들에 대하여 그림기호들을 사용자정의하는데 사용할수 있는 QFileDialog를 구축하는 수단이 있다. 그러자면 QFileIconProvider를 창조하여 그림기호들을 제공하고 대화칸에 그것을 련결한다. 다음 실례는 일정한 확장자를 가진 파일들에 대하여 사용자정의그림기호들을 정의하는 방법을 설명한다. 그림 13-15에서는 .png, .o 및 .cpp로 끝나는 파일들에 대하여 선택된 그림기호들을 보여준다.
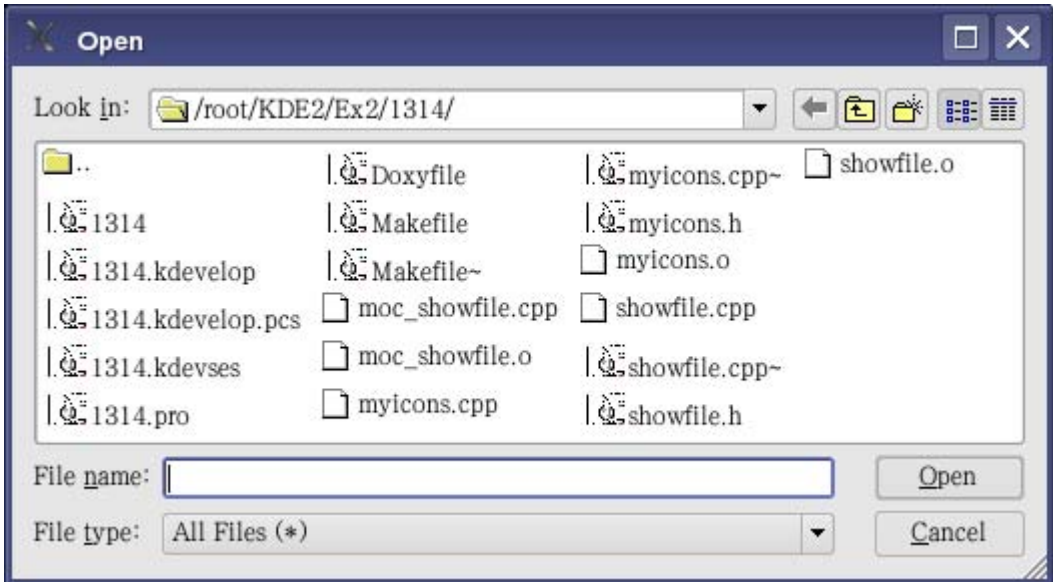
그림 13-15. 파일형을 가리키는 사용자정의그림기호들

MyIcons머리부파일

```
1 /* myicons.h */
2 #ifndef MYICONS_H
3 #define MYICONS_H
```

```
4
5 #include <qfiledialog.h>
6
7 class MyIcons: public QFileIconProvider
8 {
9 public:
10     MyIcons(QWidget *parent=0,const char *name=0);
11     ~MyIcons();
12     const QPixmap *pixmap(const QFileInfo &);
13     const QPixmap *pixmap(const QUrlInfo &);
14 private:
15     const QPixmap *selectPixmap(QString &);
16 private:
17     QPixmap *cppPixmap;
18     QPixmap *oPixmap;
19     QPixmap *pngPixmap;
20     QPixmap *filePixmap;
21     QPixmap *directoryPixmap;
22 };
23
24 #endif
```

QFileIconProvider기초클라스를 계승하는 MyIcons클라스는 다른 파일형들을 가리키는데 쓰이는 모든 픽스매프들을 가지고있다. 12행과 13행에서 선언된 pixmap()라는 이름의 2가지 메쏘드는 기초클라스의 가상메쏘드들을 재정의한다. 이 2개 메쏘드는 QFileDialog에 의하여 파일에 적합한 픽스매프를 얻는데 사용된다.

MyIcons

```
1 /* myicons.cpp */
2 #include <qfiledialog.h>
3 #include "myicons.h"
4
5 static const char *file_xpm[]={
6     "22 22 6 1",
7     " c Gray0",
8     ". c Gray51",
9     "X c Gray65",
```

```
10    "o c #dfdfdf",
11    "O c Gray100",
12    "+  c None",
13    "+ + + + + + + + + + + + + + + + + + + + + + +",
14    "+ + + + + + + + + + + + + + + + + + + + + + +",
15    "+ + + + + + + + + + + + + + + + + + + + + + +",
16    "+ + +              + + + + + + + + + + + +",
17    "+ + +  OOOOOOo + + + + + + + + + + +",
18    "+ + +  OOOOOO+ o + + + + + + + + + +",
19    "+ + +  OOOOOO+ Oo + + + + + + + + +",
20    "+ + +  OOOOOO+       + + + + + + + +",
21    "+ + +  OOOOOOOOO+  + + + + + + + +",
22    "+ + +  OOOOOOO       + + + + + + + +",
23    "+ + +  OOOOOO .+ + . + + + + + + +",
24    "+ + +  OOOOO .XX+ .. + + + + + +",
25    "+ + +  OOOOO + X+ + .+  + + + + + +",
26    "+ + +  OOOOO + + + + .+  + + + + + +",
27    "+ + +  OOOOO .+ + X.. + + + + + +",
28    "+ + +  OOOOOO .+ + . .+ + + + + +",
29    "+ + +  OOOOOOO       .    + + + + +",
30    "+ + +  OOOOOOOOOO +      + + + +",
31    "+ + +                      + +    + + + +",
32    "+ + + + + + + + + + + + + + + + + + + + + + +",
33    "+ + + + + + + + + + + + + + + + + + + + + + +",
34    "+ + + + + + + + + + + + + + + + + + + + + + +"
35 };
36
37 static const char *directory_xpm[]={
38    "15 15 6 1",
39    ". c None",
40    "b c #ffff00",
41    "d c #000000",
42    "* c #999999",
43    "a c #cccccc",
44    "c c #ffffff",
```

```
45    "..............",
46    "..*****........",
47    ".*ababa*.......",
48    "*abababa******.",
49    "*cccccccccccc*d",
50    "*cbababababab*d",
51    "*cabababababa*d",
52    "*cbababababab*d",
53    "*cabababababa*d",
54    "*cbababababab*d",
55    "*cabababababa*d",
56    "*cbababababab*d",
57    "*************d",
58    ".dddddddddddddd",
59    ".............."};
60
61 MyIcons::MyIcons(QWidget *parent,const char *name)
62    : QFileIconProvider(parent,name)
63 {
64    cppPixmap = new QPixmap("idea.png");
65    oPixmap = new QPixmap("up.png");
66    pngPixmap = new QPixmap("flag.png");
67    filePixmap = new QPixmap(file_xpm);
68    directoryPixmap = new QPixmap(directory_xpm);
69 }
70 MyIcons::~MyIcons()
71 {
72    delete cppPixmap;
73    delete oPixmap;
74    delete pngPixmap;
75    delete filePixmap;
76    delete directoryPixmap;
77 }
78 const QPixmap *MyIcons::pixmap(const QFileInfo &inf)
79 {
```

```
80    QString name = inf.fileName();
81    const QPixmap *qpixmap = selectPixmap(name);
82    if(qpixmap == NULL) {
83            if(inf.isDir())
84                    return  (directoryPixmap);
85    else
86            return(filePixmap);
87    }
88    return(qpixmap);
89 }
90 const QPixmap *MyIcons::pixmap(const QUrlInfo &inf)
91 {
92    QString name = inf.name();
93    const QPixmap *qpixmap = selectPixmap(name);
94    if(qpixmap == NULL) {
95            if(inf.isDir())
96                    return  (directoryPixmap);
97    else
98            return(filePixmap);
99    }
100   return  (qpixmap);
101 }
102 const QPixmap *MyIcons::selectPixmap(QString &name)
103 {
104   if(name.right(4) == ".cpp")
105           return  (cppPixmap);
106   if(name.right(2) == ".o")
107           return (oPixmap);
108   if(name.right(4) == ".png")
109           return  (pngPixmap);
110   return (NULL);
111 }
```

61행에서 시작하는 구성자는 XPM정보를 사용하여 픽스매프모임을 만들고 파일이름들과 련결한다. 2개 픽스매프는 5행과 37행에서 XPM자료로 정의된다. 이것들은 기정픽스매프들로서 특정한 픽스매프가 파일에 할당되지 않을 때마다 사용된다.

78행과 90행에서 선언된 pixmap()이름의 2가지 메쏘드는 파일과 련결하여 현시해야 할 픽스매프를 결정하기 위하여 파일의 설명과 함께 호출된다. 2개 메쏘드는 꼭같은 일을 수행하지만 좀 다른 인수들을 받아들인다. 81행과 93행에서 selectPixmap()호출은 파일용픽스매프를 선택하는데 사용되지만 픽스매프는 selectPixmap()로부터 얻지 않으면 두 기정픽스매프들중 하나가 선택된다.

102행의 메쏘드 selectPixmap()은 파일이름들을 시험하고 픽스매프가 할당되였는가를 결정한다. 이 실례는 간단히 파일이름을 고찰하지만 검사는 파일형을 결정하기 위하여 파일에 포함된 식별번호를 검사할수도 있다.

ShowFile머리부파일

```
1 /* showfile.h */
2 #ifndef SHOWFILE_H
3 #define SHOWFILE_H
4
5 #include <qwidget.h>
6 #include <qlabel.h>
7 #include <qstring.h>
8
9 class ShowFile: public QWidget
10 {
11     Q_OBJECT
12 public:
13     ShowFile(QWidget *parent=0,const char *name=0);
14 private:
15     QLabel *filelabel;
16     QString filename;
17 private slots:
18     void popupOpen();
19 };
20
21 #endif
```

ShowFile

```
1 /* showfile.cpp */
2 #include <kapplication.h>
3 #include <qpushbutton.h>
4 #include <qlayout.h>
```

```
5 #include <qfiledialog.h>
6 #include "showfile.h"
7 #include "myicons.h"
8
9 int main(int argc,char **argv)
10 {
11     KApplication app(argc,argv,  "showfile");
12     QFileIconProvider *provider = new MyIcons();
13     QFileDialog::setIconProvider(provider);
14     ShowFile showfile;
15     showfile.show();
16     app.setMainWidget(&showfile);
17     return  (app.exec());
18 }
19 ShowFile::ShowFile(QWidget *parent,const char *name)
20     : QWidget(parent,name)
21 {
22     QPushButton *button;
23     QVBoxLayout *box = new QVBoxLayout(this,0,3);
24
25     filelabel = new QLabel("",this);
26     filelabel->setAlignment(Qt::AlignHCenter);
27     box->addWidget(filelabel);
28
29     button = new QPushButton("Select File to Open",this);
30     box->addWidget(button);
31     connect(button,SIGNAL(clicked()),
32             this,SLOT(popupOpen()));
33
34     resize(10,10);
35     box->activate();
36 }
37 void ShowFile::popupOpen()
38 {
39     QString name = QFileDialog::getOpenFileName("",
```

```
40            NULL,this);
41    if(!name.isEmpty()) {
42            filename = name;
43            filelabel->setText(filename);
44    }
45 }
```

이 프로그람은 파일대화칸과 그림기호제공기를 련결하고 대화칸을 펼치는데 쓰이는 단추들을 제공한다.

그림기호제공기는 12행에서 창조된다. 13행의 정적메쏘드 setIconProvider()호출은 모든 QFileDialog객체에 대하여 그림기호제공기로서 MyIcon객체를 할당한다. 이 기구는 기정그림기호제공기를 새로운것으로 바꾼다. 이 기술을 사용하여 자기가 좋아하는 많은 그림기호제공기들을 가질수 있고 그것들을 변경할수 있다.

단추를 찰칵할 때마다 17행의 처리부메쏘드 popupOpen()가 호출된다. QFileDialog창문은 39행의 getOpenFilename()호출에 의해 펼쳐진다. 41행은 파일이름이 선택되였는가 결정하고 선택되였으면 파일이름을 현시한다.

요 약

이 장은 매우 특수한 도형조작에 대하여 설명하였다. KDE와 Qt API의 부분으로 포함된 수단들은 도형화상에 대한 조작을 가능하게 한다.

- 창문에 화소들을 그리는데 사용하는 API는 인쇄페지에 화소들을 그리는데도 사용할수 있다.
- 도형화상, 또는 그 일부분을 만드는데 요구되는 단계별명령을 기억하고 임의의 회수 재생할수 있다.
- 도형객체는 원래보다 더 크거나 작은 창문에 맞게 확대축소할수 있다.
- 도형객체에 대하여 비례확대축소, 잘라내기, 경사지기, 변환, 그리고 회전을 비롯한 많은 조작을 수행할수 있다.
- 동화는 도형프레임들의 렬과 시계를 사용하여 만들수 있다.

제14장. 끌어다놓기

학습내용
- 응용프로그람안에서 본문을 끌어다놓기
- 응용프로그람들중에 본문과 도형의 끌어다놓기
- 체계오려둠판을 리용한 도형의 자르기와 붙이기

표준자료전송능력은 서로 다른 두 응용프로그람들이 통합되는것처럼 사용자에게 보이도록 할수 있다. 이 결합은 일반적으로 2가지 방법으로 진행한다. 마우스를 사용하여 한 창문에서 다른 창문에로 도형객체를 끌고감으로써 자료를 한 응용프로그람에서 다른 응용프로그람까지 이동시킬수 있다. 다른 수법은 사용자가 체계오려둠판에 자료를 복사하고 다른 응용프로그람이 오려둠판으로부터 자료를 읽어들일수 있게 하는것이다.

끌어다놓기는 응용프로그람들사이의 통신에 매우 쓸모있고 또한 하나의 응용프로그람에서 진행한 조작에도 아주 쓸모있다. 사용자는 응용프로그람안에서 한 폼으로부터 다른 폼에로 객체를 이동할수 있고 혹은 하나의 창문안에서 객체들의 위치를 바꿀수 있다.

이 장은 끌어다놓기조작에서 발생하는 사건들에 대하여 설명한다. 응용프로그람은 끌기조작이 요구되였다는것을 인식하고 자료를 끌기용으로 묶어야 하며 놓기목표는 놓기가 발생하였다는것을 인식하고 자료의 묶음을 열고 분배하여야 한다.

제1절. 본문끌어다놓기

다음 프로그람은 한 위치에서 다른 위치에로 본문의 끌어다놓기를 실현한다. 임의의 창문부품도 끌기조작의 원천, 놓기조작의 목표, 또는 두가지로서 작용할수 있다. 프로그람의 끌기와 놓기조작들은 모두 원천과 목표창문부품들에 의해 직접 조종된다.

DragDrop머리부파일

```
1 /* dragdrop.h */
2 #ifndef DRAGDROP_H
3 #define DRAGDROP_H
4
5 #include <qwidget.h>
6 #include <qstring.h>
7 #include "dragfrom.h"
8 #include "dropto.h"
9
10 class DragDrop: public QWidget
11 {
12 public:
```

```
13     DragDrop(QWidget *parent=0,const char *name=0);
14 private:
15     DragFrom *apples;
16     DragFrom *oranges;
17     DropTo *target;
18 };
19
20 #endif
```

15행과 16행에서 2개 창문부품 apples와 oranges는 본문끌기조작의 원천으로서 동작하며 창문부품 target는 본문놓기조작의 목표로서 리용된다.

DragDrop

```
1 /* dragdrop.cpp */
2 #include <kapplication.h>
3 #include <qlayout.h>
4 #include <qlabel.h>
5 #include "dragdrop.h"
6
7 int main(int argc,char **argv)
8 {
9     KApplication app(argc,argv,  "dragdrop");
10    DragDrop dragdrop;
11    dragdrop.show();
12    app.setMainWidget(&dragdrop);
13    return  (app.exec());
14 }
15
16 DragDrop::DragDrop(QWidget *parent,const char *name)
17    : QWidget(parent,name)
18 {
19    QVBoxLayout *box = new QVBoxLayout(this,30);
20    box->addSpacing(30);
21
22    target = new DropTo("target",this);
23    box->addWidget(target);
24
```

```
25    apples = new DragFrom("apples",this);
26    box->addWidget(apples);
27
28    oranges = new DragFrom("oranges",this);
29    box->addWidget(oranges);
30
31    box->activate();
32 }
```

DragDrop창문은 끌어다놓기가 가능한 3개 창문부품의 기본창문이다. 놓기조작의 목표는 22행과 23행에서 수직칸배치에 추가된다. 2개 본문끌기원천창문부품은 25∼29행에서 기본창문에 추가된다. 그림 14-1은 apples창문부품의 본문을 목표에 끌어다놓은 후에 창문을 보여준다.
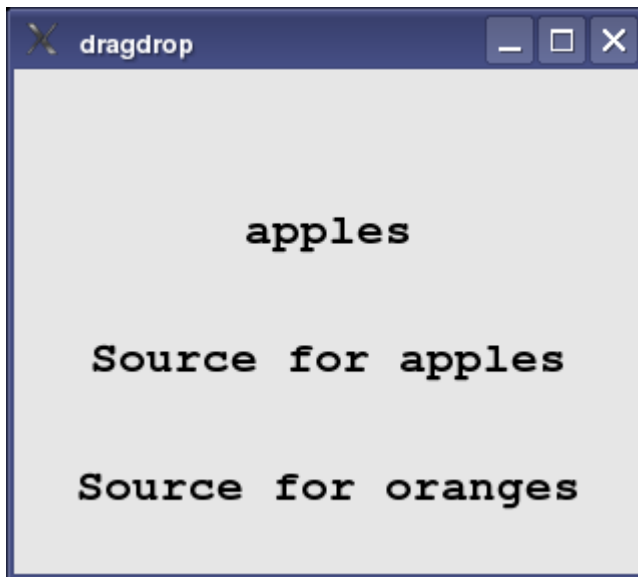
그림 14-1. 아래에서 우로 본문의 끌어다놓기

DragFrom머리부파일

```
1 /* dragfrom.h */
2 #ifndef DRAGFROM_H
3 #define DRAGFROM_H
4
5 #include <qlabel.h>
6 #include <qstring.h>
7
8 class DragFrom: public QLabel
9 {
10 public:
```

```
11     DragFrom(const char *text,QWidget *parent=0);
12 private:
13     QString string;
14 protected:
15     virtual void mousePressEvent(QMouseEvent *);
16 };
17
18 #endif
```

DragFrom

```
1 /* dragfrom.cpp */
2 #include <qlabel.h>
3 #include <qfont.h>
4 #include <qdragobject.h>
5 #include "dragfrom.h"
6
7 DragFrom::DragFrom(const char *text,QWidget *parent)
8      : QLabel(parent)
9 {
10     string = text;
11     QString label("Source for ");
12     label.append(text);
13     setText(label);
14     setAlignment(Qt::AlignHCenter);
15     QFont font("Courier",18,QFont::Bold,FALSE);
16     setFont(font);
17 }
18 void DragFrom::mousePressEvent(QMouseEvent *)
19 {
20 QDragObject *textdrag = new QTextDrag(string,this);
21 textdrag->dragCopy();
22 }
```

DragFrom창문부품은 기초클라스로서 QLabel창문부품을 사용한다. 7행에서 구성자에 제공된 본문은 10행에서 문자렬에 보관되여 이 창문부품으로부터 끌수 있는 본문이며 한편 현시된 본문은 11～13행에서 “Source for”에 의해서 선행된다. 14～16행은 본문을 중심에 배치하고 서체를 지정한다.

메쏘드 mousePressEvent()는 마우스단추를 누를 때마다 호출되며 QTextDrag객체는 20행에서 만들어진다. dragCopy()호출은 QTextDrag객체가 마우스에 따라 이동하게 한다. 그 객체는 끌기 및 놓기조작(마우스단추를 놓을 때 그것이 목적지에 이르렀는가 아닌가)에 의해 삭제되므로 자기 프로그람에서 그것을 삭제하지 말아야 한다. 이 프로그람은 임의의 수의 QTextDrag객체들을 창조하고 보낼수 있지만 그것들중 어떤 객체의 마지막 처리에 대하여 절대로 통지되지 않는다.

DropTo머리부파일

```
1 /* dropto.h */
2 #ifndef DROPTO_H
3 #define DROPTO_H
4
5 #include <qlabel.h>
6 #include <qevent.h>
7 #include <qstring.h>
8
9 class DropTo: public QLabel
10 {
11 public:
12     DropTo(const char *text,QWidget *parent=0);
13 protected:
14     void dragEnterEvent(QDragEnterEvent *e);
15     void dropEvent(QDropEvent *e);
16 };
17
18 #endif
```

메쏘드 dragEnterEvent()와 dropEvent()는 QWidget기초클라스의 메쏘드선언들을 재정의한다. 이 메쏘드들은 끌어다놓기조작에서 마우스를 놓을 위치를 찾고 있을 때 호출된다.

DropTo

```
1 /* dropto.cpp */
2 #include <qlabel.h>
3 #include <qfont.h>
4 #include <qdragobject.h>
5 #include "dropto.h"
6
7 DropTo::DropTo(const char *text,QWidget *parent)
```

```
8      : QLabel(text,parent)
9 {
10     setAlignment(Qt::AlignHCenter);
11     QFont font("Courier",18,QFont::Bold,FALSE);
12     setFont(font);
13     setAcceptDrops(TRUE);
14 }
15 void DropTo::dragEnterEvent(QDragEnterEvent *e)
16 {
17     e->accept(QTextDrag::canDecode(e));
18 }
19 void DropTo::dropEvent(QDropEvent *e)
20 {
21     QString text;
22
23     if(QTextDrag::decode(e,text))
24             setText(text);
25 }
```

DropTo클라스의 기초클라스는 QLabel이며 거기에 놓는 본문을 현시한다. 구성자중 10～12행은 현시된 본문의 배치와 서체를 지정한다.

13행의 setAcceptDrops()호출은 마우스끌어다놓기조작의 목표로서 이 창문부품이 동작하게 한다. 마우스가 끌기조작을 수행하고 있고 마우스지시기가 이 창문부품에 들어갈 때마다 메쏘드 dragEnterEvent()는 끌고있는 자료에 대한 정보와 함께 호출된다. 또한 자료가 이 창문부품안에 떨어지면 dropEvent()가 호출된다.

15행의 dragEnterEvent()는 마우스끌기가 이 창문부품의 경계에 들어갈 때마다 호출된다. 이 메쏘드의 목적은 창문부품이 놓기를 받아들이려고 하는가를 판단하는것이다. 이 실례는 간단히 끌고있는 자료를 본문으로 변환할수 있는가를 판단하기 위하여 QTextDrag클라스의 정적canDecode()메쏘드를 호출한다. 만일 변환할수 있다면 TRUE인수와 함께 accept()호출이 이루어지고 그렇지 않으면 FALSE인수와 함께 호출된다.

19행의 메쏘드 dropEvent()는 일정한 사건렬이 발생하면 호출된다. 마우스의 끌어다놓기조작이 있고 이 창문부품에 들어갔으며 dragEnterEvent()메쏘드가 사건의 accept()메쏘드를 TRUE를 가지고 호출하였으면 dropEvent()메쏘드가 호출된다. 이 실례는 본문을 받아들이고 23행은 QTextDrag클라스의 정적decode()메쏘드호출을 만들며 결과복호화된 본문이 표식자의 본문설정에 사용되고 끌어다놓기조작은 성공적으로 완료된다.

제2절. 본문과 화상자료의 끌어다놓기

다음 2개 프로그람은 한 응용프로그람에서 다른 응용프로그람에로 객체들을 끌고가는 방법과 받아들이는 프로그람이 놓은 자료의 형을 판단하는 능력을 보여준다. 2개의 끌기원천 즉 본문과 화상원천이 있으나 오직 하나의 놓기목표가 있다. 놓기목표는 들어오는 자료형을 판단하고 그에 따라 작용한다.

DateImage머리부파일

```
1 /* dateimage.h */
2 #ifndef DATEIMAGE_H
3 #define DATEIMAGE_H
4
5 #include <qwidget.h>
6 #include <qlabel.h>
7 #include <qdragobject.h>
8
9 class DateImage: public QWidget
10 {
11 public:
12     DateImage(QWidget *parent=0,const char *name=0);
13 };
14
15 class DateSource: public QLabel
16 {
17 public:
18     DateSource(QWidget *parent=0);
19 protected:
20     virtual void mousePressEvent(QMouseEvent *);
21 };
22
23 class ImageSource: public QLabel
24 {
25 public:
26     ImageSource(QWidget *parent=0);
27 protected:
28     virtual void mousePressEvent(QMouseEvent *);
```

```
29 };
30
31 #endif
```

이 머리부파일은 끌고있는 자료의 원천인 클라스들의 선언을 포함한다. DateImage클라스는 현시될 때 DateSource객체와 ImageSource객체 모두를 포함하는 제일웃준위창문이다. DateSource와 ImageSource클라스는 모두 QLabel의 파생클라스이므로 그것들은 모두 본문을 현시하고 끌어다놓기조작의 원천으로 사용될수 있다. DateSource로부터 끌기되는 자료는 본문문자렬이고 ImageSource로부터 끌기되는 자료는 QImage객체이다.

DateImage

```
1 /* dateimage.cpp */
2 #include <kapplication.h>
3 #include <qlayout.h>
4 #include <qlabel.h>
5 #include <qfont.h>
6 #include <time.h>
7 #include "dateimage.h"
8
9 #include "bluemarble.xpm"
10
11 int main(int argc,char **argv)
12 {
13    KApplication app(argc,argv,  "dateimage");
14    DateImage dateimage;
15    dateimage.show();
16    app.setMainWidget(&dateimage);
17    return  (app.exec());
18 }
19
20 DateImage::DateImage(QWidget *parent,const char *name)
21    : QWidget(parent,name)
22 {
23    QVBoxLayout *box = new QVBoxLayout(this,30);
24
25    DateSource *ds = new DateSource(this);
26    box->addWidget(ds);
```

```
27
28    ImageSource *is = new ImageSource(this);
29    box->addWidget(is);
30
31    box->activate();
32 }
33
34 DateSource::DateSource(QWidget *parent)
35    : QLabel("Date",parent)
36 {
37    setAlignment(Qt::AlignHCenter);
38    QFont font("Courier",18,QFont::Bold,FALSE);
39    setFont(font);
40 }
41 void DateSource::mousePressEvent(QMouseEvent *)
42 {
43    time_t t;
44    char *ct;
45
46    t = time((time_t *)0);
47    ct = ctime(&t);
48    QString string(ct);
49    QDragObject *textdrag = new QTextDrag(string,this);
50    textdrag->dragCopy();
51 }
52
53
54 ImageSource::ImageSource(QWidget *parent)
55    : QLabel("Image",parent)
56 {
57    setAlignment(Qt::AlignHCenter);
58    QFont font("Courier",18,QFont::Bold,FALSE);
59    setFont(font);
60 }
61 void ImageSource::mousePressEvent(QMouseEvent *)
```

```
62 {
63     QImage image(magick);
64     QDragObject *imagedrag = new QImageDrag(image,this);
65     imagedrag->dragCopy();
66 }
```

20행의 DateImage구성자는 수직칸을 사용하여 DateSource객체와 ImageSource객체를 포함한다. DateSource와 ImageSource는 모두 기초클라스로서 QLabel을 사용하며 그림 14-2에서 보여준것과 같은 창문을 만든다.

그림 14-2. 본문과 화상끌기조작의 원천

34행의 DateSource구성자는 그 자체의 서체와 본문배치를 설정한다. 35행에서 QLabel기초클라스의 초기화는 현시본문으로서 "Date"를 지정한다. 41행의 mousePressEvent()는 현재체계시간을 포함하는 문자렬을 만들고 그것을 리용하여 49행에서 QTextDrag객체를 구성한다. 50행의 dragCopy()호출은 QTextDrag객체를 마우스에 련결하고 그것을 그 방향으로 보낸다.

54행의 ImageSource구성자는 그 자체의 서체와 본문배치를 설정한다. 55행의 QLabel기초클라스초기화는 현시본문으로서 "Image"를 지정한다. 61행의 mousePressEvent()는 9행에서 포함한 XPM자료로부터 QImage객체를 만든다. QImage객체는 64행에서 QImageDrag객체의 창조에 사용된다. 65행의 dragCopy()호출은 QImageDrag객체를 끌수 있도록 마우스에 련결한다.

Target머리부파일

```
1 /* target.h */
2 #ifndef TARGET_H
3 #define TARGET_H
4
5 #include <qwidget.h>
6 #include "target.h"
7
8 class Target: public QWidget
```

```
9 {
10 public:
11     Target(QWidget *parent=0,const char *name=0);
12 protected:
13     void dragEnterEvent(QDragEnterEvent *e);
14     void dropEvent(QDropEvent *e);
15 };
16
17 #endif
```

놓기목표는 창문부품의 제일웃준위창문이다. 이것을 달성하기 위하여 QWidget가상메쏘드 dragEnterEvent()와 dropEvent()를 재정의해야 한다.

Target

```
1 /* target.cpp */
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include <qdragobject.h>
5 #include "target.h"
6
7 int main(int argc,char **argv)
8 {
9     KApplication app(argc,argv,  "target");
10    Target target;
11    target.show();
12    app.setMainWidget(&target);
13    return  (app.exec());
14 }
15
16 Target::Target(QWidget *parent,const char *name)
17    : QWidget(parent,name)
18 {
19    setFixedSize(400,300);
20    setAcceptDrops(TRUE);
21 }
22 void Target::dragEnterEvent(QDragEnterEvent *e)
23 {
```

```
24    e->accept(QTextDrag::canDecode(e) ||
25    QImageDrag::canDecode(e));
26 }
27 void Target::dropEvent(QDropEvent *e)
28 {
29    QString text;
30    QImage image;
31
32    if(QTextDrag::decode(e,text)) {
33          drawText(e->pos(),text);
34    }
35    if(QImageDrag::decode(e,image)) {
36          QPainter *p = new QPainter(this);
37          p->drawImage(e->pos(),image);
38    }
39 }
```

16행의 구성자는 기본창문의 크기를 고정시키고 setAcceptDrops()를 호출하여 이 창문을 놓기목표로서 가능하게 한다.

22행의 dragEnterEvent()는 끌기하는 마우스가 이 창문의 경계에 들어갈 때마다 호출된다. 끌고있는 자료형을 창문이 받아들일수 있으면 QDragEnterEvent객체의 accept()메쏘드에 TRUE를 넘기여 호출하고 그렇지 않으면 FALSE로 호출한다. 이 창문은 본문과 화상자료 모두 받아들일수 있으므로 이것들중 어느것이나 복호화할수 있을 때 TRUE로 된다.

마우스로 이 창문부품우에 자료를 놓으면 27행에서 dropEvent()가 호출된다. 32행에서 QTextDrag의 decode()호출은 자료를 얻으려고 한다. 복호화가 성공하면 자료는 본문에 기억되고 33행의 drawText()호출은 창문에 본문을 그린다. 국부창문에서 본문의 정확한 위치는 마우스지시자를 놓는 위치에 의해 결정되며 그 위치는 QDropEvent객체의 pos()호출로부터 QPoint객체로서 얻는다. 복호화절차는 QImageDrag클라스의 decode()메쏘드를 호출함으로써 반복된다. 화상이 성공적으로 복호화되면 QPainter객체는 놓기위치에서 창문에 화상을 그리는데 사용된다. 그림 14-3에서는 몇개의 본문과 화상을 놓은 후에 목표창문을 보여준다.
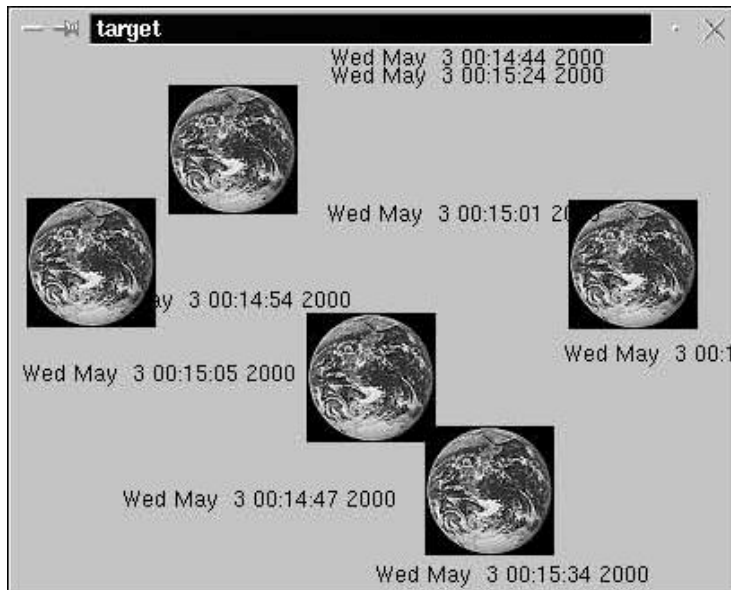
그림 14-3. 같은 창문에 놓인 본문과 화상자료

2가지 다른 방법은 특별한 도형을 현시하려고 하는 경우에 놓기목표창문부품에 의해 마음대로 사용될수 있다. 다음 메쏘드는 끌고있는 마우스가 아무것도 놓지 않고 창문을 떠날 때마다 호출된다.

void Target::dragExitEvent(QDragExitEvent *e)

이 메쏘드를 리용하여 사용자에게 더 많은 반결합을 제공할수 있다. 실례로 이 메쏘드는 dragEnterEvent()와 결합하여 끌고있는 마우스가 창문우로 날아다닐 때마다 목표창문부품을 강조한다. 이것은 몇가지 작은 목표창문들이 서로 옆에 있을 때 리용할수 있다. 마우스가 목표창문안에서 위치를 바꿀 때마다 호출되는 다음의 메쏘드를 리용하여 가능한 놓기위치를 추적할수 있다.

void Target::dragMoveEvent(QDragMoveEvent *e)

이 메쏘드를 사용하는 하나의 실례는 +기호 혹은 다른 지시기를 가능한 놓기위치에 배치하는것이다.

제3절. 자르기와 붙이기

끌어다놓기에 사용한것과 같은 기초기구를 자르기와 붙이기에 사용할수 있다. 객체를 한 응용프로그람에서 다른 응용프로그람에로 끌기하는것은 그것을 오려둠판에 복사하는것과 같은데 차이나는것은 대면부가 바뀐다는것이다.

다음 프로그람은 자르기, 복사 및 붙이기조작을 리용하여 그자체와 오려둠판사이에서 이동할수 있다. 그림 14-4에서는 KDE오려둠판과 프로그람사이에서 두방향으로 픽스매프를 전송하는데 사용하는 창문과 단추들을 보여준다.
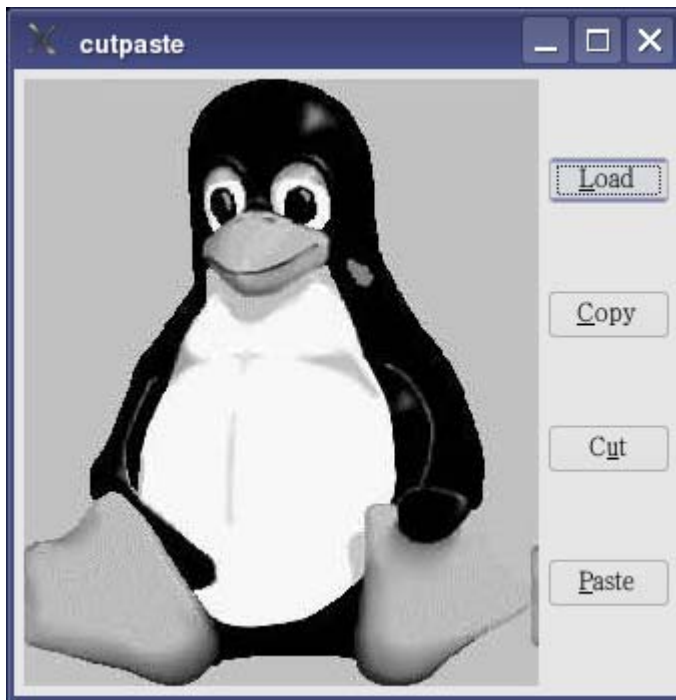
그림 14-4. KDE오려둡판과 프로그람사이에서 량방향으로 화상의 복사

CutPaste머리부파일

```
1 /* cutpaste.h */
2 #ifndef CUTPASTE_H
3 #define CUTPASTE_H
4
5 #include <qwidget.h>
6 #include <qpixmap.h>
7
8 class CutPaste: public QWidget
9 {
10     Q_OBJECT
11 public:
12     CutPaste(QWidget *parent=0,const char *name=0);
13 private:
14     QWidget *widget;
15     QPixmap *pixmap;
16 private slots:
17     void loadButton();
18     void copyButton();
```

```
19    void cutButton();
20    void pasteButton();
21 };
22
23 #endif
```

14행의 QWidget는 QPixmap를 현시하는데 사용된다. 15행의 QPixmap는 현재 현시되는 화상이다. 4가지 처리부메쏘드는 누름단추들에 응답하는것들이다.

CutPaste

```
1 /* cutpaste.cpp */
2 #include <kapplication.h>
3 #include <qlayout.h>
4 #include <qimage.h>
5 #include <qdragobject.h>
6 #include <qclipboard.h>
7 #include <qpushbutton.h>
8 #include "cutpaste.h"
9
10 int main(int argc,char **argv)
11 {
12    KApplication app(argc,argv,  "cutpaste");
13    CutPaste *cutpaste = new CutPaste();
14    cutpaste->show();
15    app.setMainWidget(cutpaste);
16    return  (app.exec());
17 }
18
19 CutPaste::CutPaste(QWidget *parent,const char *name)
20    : QWidget(parent,name)
21 {
22    QPushButton *button;
23    QHBoxLayout *hlayout = new QHBoxLayout(this,5);
24    QVBoxLayout *vlayout = new QVBoxLayout();
25
26    pixmap = NULL;
27
```

```
28    button = new QPushButton("Load",this);
29    connect(button,SIGNAL(clicked()),
30            this,SLOT(loadButton()));
31    vlayout->addWidget(button);
32
33    button = new QPushButton("Copy",this);
34    connect(button,SIGNAL(clicked()),
35            this,SLOT(copyButton()));
36    vlayout->addWidget(button);
37
38    button = new QPushButton("Cut",this);
39    connect(button,SIGNAL(clicked()),
40            this,SLOT(cutButton()));
41    vlayout->addWidget(button);
42
43    button = new QPushButton("Paste",this);
44    connect(button,SIGNAL(clicked()),
45            this,SLOT(pasteButton()));
46    vlayout->addWidget(button);
47
48    widget = new QWidget(this);
49    widget->setFixedSize(257,303);
50    widget->setBackgroundColor(QColor("white"));
51
52    hlayout->addWidget(widget);
53    hlayout->addLayout(vlayout);
54
55    resize(10,10);
56    hlayout->activate();
57 }
58 void CutPaste::loadButton()
59 {
60    if(pixmap != NULL)
61            delete pixmap;
62    pixmap = new QPixmap("logo.xpm");
```

```
63    widget->setBackgroundPixmap(*pixmap);
64 }
65 void CutPaste::copyButton()
66 {
67    if(pixmap != NULL) {
68            QImage image = pixmap->convertToImage();
69            QDragObject *drag = new QImageDrag(image,this);
70            QClipboard *clipboard = QApplication::clipboard();
71            clipboard->setData(drag);
72    }
73 }
74 void CutPaste::cutButton()
75 {
76    if(pixmap != NULL) {
77            copyButton();
78            widget->setBackgroundColor(QColor("white"));
79            delete pixmap;
80            pixmap = NULL;
81    }
82 }
83 void CutPaste::pasteButton()
84 {
85    QClipboard *clipboard = QApplication::clipboard();
86    QMimeSource *mime = clipboard->data();
87    QImage image;
88    if(QImageDrag::decode(mime,image)) {
89            QPixmap *newPixmap = new QPixmap();
90            if(newPixmap->convertFromImage(image)) {
91                    if(pixmap != NULL)
92                            delete pixmap;
93                    pixmap = newPixmap;
94                    widget->setBackgroundPixmap(*pixmap);
95            }
96    }
97 }
```

19행에서 시작하는 구성자는 자료를 초기화하고 수직칸에 단추모임을 삽입하여 현시하며 수평칸에 수직칸과 창문부품을 삽입한다. 초기픽스매프는 현시되지 않으므로 26행에서 NULL로 초기화된다. 도형현시창문부품은 48행에서 만들어지고 고정백색배경으로 초기화된다.

58행의 처리부메쏘드 loadButton()은 파일로부터 새로운 픽스매프를 적재한다. 60행과 61행은 이전에 존재하는 픽스매프를 삭제하고 63행의 setBackgroundPixmap()호출은 새로 적재한 픽스매프를 현시한다.

65행의 처리부메쏘드 copyButton()은 픽스매프가 존재하는가 검사하고 존재하면 그것을 오려둠판에 복사한다. 68행의 convertToImage()호출은 픽스매프를 QImage로 변환한다. 왜냐하면 그것이 QImageDrag에 요구되는 도형이기때문이다. QClipboard객체의 주소는 70행에서 clipboard()메쏘드호출에 의해 얻고 자료는 71행에서 setData()호출에 의해 오려둠판에 보관된다.

74행의 처리부메쏘드 cutButton()는 픽스매프가 존재하는가 검사하고 존재하면 그것을 오려둠판에 복사하고 그것을 국부적으로 삭제한다. copyButton()호출은 오려둠판에 픽스매프를 복사한다. setBackgroundColor()호출은 창문으로부터 픽스매프를 지우고 79행과 80행은 기억기로부터 픽스매프를 삭제한다.

83행의 처리부메쏘드 pasteButton()는 오려둠판에서 이 응용프로그람에로 픽스매프를 읽어들인다. 85행에서 정적메쏘드 clipboard()호출은 체계오려둠판의 주소를 얻는다. 오려둠판은 자료를 QMineSource객체로 보관하며 이것은 86행의 data()호출에 의해 얻어진다. 몇가지 다른 자료형들을 오려둠판에 기억할수 있으므로 88행에서 decode()호출로부터 돌아오는 Boolean돌림값은 자료가 성공적으로 QImage객체로 변환되였는가 검사되여야 한다. 변환이 성공하면 90행의 convertFromImage()호출은 자료로부터 픽스매프를 창조하며 91～94행은 현존픽스매프를 새로운것과 교체하고 그것을 새 현시배경으로서 창문부품에 보관한다.

요 약

한 위치에서 다른 위치에로 자료를 끌고가는것, 또는 한 위치에서 자료를 자르고 다른 위치에 그것을 붙일 때 송신자와 수신자 모두가 자료형과 그것을 묶는 방법을 일치시킬것을 요구한다. 응용프로그람의 견지로부터 자료전송과 수신은 함수호출보다 훨씬 더 쉽다. 이 장은 다음과 같은 자료의 끌어다놓기에 대하여 설명하였다.

- 다른 장소로 자료를 끌고가려면 우선 QDragObject에 그것을 일봉해야 한다. 창문이 떨구어놓은 객체를 받아들이려면 QDragObject의 자료를 복호화할 준비가 되여있어야 한다.
- setAcceptDrops()는 떨구어놓은 자료를 창문부품이 받아들이기전에 호출되여야 한다.
- 자르기와 붙이기조작은 체계QClipboard객체가 자료보관에 매개물로 사용되는것외에는 끌어다놓기와 본질적으로 갈다.

다음 장은 애플레프를 론의한다. 애플레트는 KDE환경에서 기본창문의 밑(또는 다른 변)에 있는 구획에 나타나는 작은 그림기호부류의 창문이다. 또한 한 응용프로그람에서 다른 응용프로그람에로 자료를 보내는 다른 방법들을 론의한다.

제15장. 프로쎄스사이 통신과 애플레트

학습내용

지령행에 인수넘기기

한 실행프로그람으로부터 다른 실행프로그람에로 자료블로크송신

파넬에서 애플레트를 통한 사용자호출의 제공

오직 프로쎄스의 한개 실례만 임의의 시간에 실행한다는것의 담보

한 프로그람으로부터 다른 프로그람에로 자료를 넘기는데는 2가지 기본방법이 있다. 기동할 때 인수들을 지령행에 줄수 있고 실행시에 자료블로크가 한 응용프로그람에 의해 생성되여 그것을 기다리고있는 다른 프로쎄스에 넘어갈수 있다. KDE는 이 두가지 통신방법을 모두 제공한다.

지령행으로부터 오는 정보를 분석하고 보관하는 지령행클라스가 있다. 그밖에도 어느정도 응용프로그람에서 사용할수 있는 환경을 표준화하는 KDE선택기발들에 대한 호출을 제공한다. 즉 이 객체를 사용함으로써 다른 응용프로그람들은 표준모임의 기발들에 표준방법으로 응답하도록 프로그람을 작성할수 있다.

프로쎄스사이통신모형은 통보문들을 조종하기 위하여 배경에서 실행하는 봉사기를 요구한다. 이 봉사기는 우편국과 같은 기능을 수행한다. 매개 응용프로그람은 이름에 의해 식별되는 P.O.(우편국)칸을 얻으며 다른 응용프로그람들은 거기에 통보문들을 보관할수 있다.

애플레트는 기본KDE창문의 한쪽에 존재하는 KDE조종판(KDE kicker)에 그림기호로서 창문을 현시하는 특수한 응용프로그람이다. 애플레트는 그 제일웃준위창문으로서 매우 작은 창문을 가지는 결함을 가지고 있으나 항상 사용자가 볼수 있는 우점이 있다.

이 장은 자기 프로그람이 이 자료교환방법과 애플레트의 우점을 리용하는 여러가지 방법을 설명한다.

제1절. DCOP통신모형

탁상통신규약(DCOP, Desktop Communications Protocol)쏘프트웨어는 프로쎄스의 그룹 안에서 프로쎄스사이통신을 설정하는 매우 간단한 방법을 제공하기 위하여 개발되였다. 데몬프로쎄스를 통한 모든 통신을 dcop봉사기라고 부른다. 통보문을 송신 혹은 수신하려는 한개 프로쎄스는 우선 dcopserver로서 이름을 등록하고 다른 프로쎄스는 dcopserver의 보호하에 그 이름에 통보문들을 보낼수 있다.

DCOP는 실제로 RPC(원격수속호출)기계의 간단한 형식이다. 통보문은 인수를 요구하거나 요구하지 않으며 돌림값을 주거나 주지 않을수 있는 함수호출형식으로 송신된다.

다음 실례는 3개의 프로그람으로 이루어진다. wilbur프로그람은 자기를 dcopserver로 등록하고 통보문이 도착하기를 기다린다. 프로그람 tellwilbur는 wilbur에게 통보문을 송신하고 응답을 기다리지 않지만 askwilbur는 통보문을 보내고 응답을 기다린다.

Wilbur머리부파일

```
1 /* wilbur.h */
2 #ifndef WILBUR_H
3 #define WILBUR_H
4
5 #include <qmultilineedit.h>
6 #include <dcopobject.h>
7
8 class WilReceiver: public QMultiLineEdit, public DCOPObject
9 {
10 public:
11     WilReceiver(const char *name=0);
12     bool process(const QCString &function,
13             const QByteArray &data,QCString &replyType,
14             QByteArray &replyData);
15     double cubeRoot(double value);
16 private:
17 };
18
19 #endif
```

WilReceiver는 DCOPObject이므로 그것은 통보문들을 받아들이고 국부수속을 실행하고 통보문의 발신자에게 결과를 되돌려줄수 있다. 또한 QMultiLineEdit창문부품을 계승하므로 WilReceiver는 창문부품이다.

12행에서 DCOPObject클라스의 순수가상메쏘드 process()를 재정의한다. 그것은 통보문을 다른 메쏘드로부터 받아들일 때마다 호출된다. 15행의 메쏘드 cubeRoot()는 다른 메쏘드로부터 호출될수 있다.

Wilbur

```
1 /* wilbur.cpp */
2 #include <kapplication.h>
3 #include <qcstring.h>
4 #include <qmultilineedit.h>
5 #include <dcopclient.h>
6 #include <math.h>
7 #include "wilbur.h"
8
```

```
9 int main(int argc,char **argv)
10 {
11     QString str;
12     KApplication app(argc,argv, "wilbur");
13
14     DCOPClient *client = app.dcopClient();
15     QCString dcopID = client->registerAs(app.name(), FALSE);
16
17     WilReceiver *wilbur = new WilReceiver("wilreceiver");
18     app.setMainWidget(wilbur);
19
20     str.sprintf("wilbur registered as \"%s\"",
21             dcopID.data());
22     wilbur->insertLine(str);
23
24     int returnValue = app.exec();
25     client->detach();
26     return  (returnValue);
27 }
28 WilReceiver::WilReceiver(const char *name)
29     : DCOPObject(name)
30 {
31     setReadOnly(TRUE);
32     show();
33 }
34 bool WilReceiver::process(const QCString &function,
35     const QByteArray &data,
36     QCString &replyType,
37     QByteArray &replyData)
38 {
39     if(function == "cubeRoot(double)") {
40             double inValue;
41             double outValue;
42             QDataStream inStream(data,IO_ReadOnly);
43             inStream >> inValue;
```

```
44          outValue = cubeRoot(inValue);
45          QDataStream outStream(replyData,IO_WriteOnly);
46          outStream << outValue;
47          replyType = "double";
48          return  (TRUE);
49      } else {
50          QString string;
51          string.sprintf("call to unknown function %s",
52          function.data());
53          insertLine(string);
54          return  (FALSE);
55      }
56 }
57 double WilReceiver::cubeRoot(double value)
58 {
59      QString string;
60      double root = cbrt(value);
61      string.sprintf("Cube root of %g is %g",value,root);
62      insertLine(string);
63      return (root);
64 }
```

이 프로그람은 제일웃준위창문부품으로서 WilReceiver객체를 사용한다. 이것은 본문을 현시하며 들어오는 통보문들에 응답할 능력에 있다.

dcopserver를 통하여 통신하는 모든 프로쎄스는 의뢰기로 등록되여야 한다. 14행의 dcopClient()호출은 국부DCOPClient객체를 만들고 그 주소를 돌려준다. 15행의 registerAs()호출은 dcopserver데몬과 의뢰기의 이름을 등록한다. 12행에서 지정한 응용프로그람의 이름은 "Wilbur"이므로 지금까지 "wilbur"에 보내온 통보문은 이 프로그람에 되돌아온다. 실제 등록이름은 15행의 dcopID안의 문자렬로서 보관된 돌림값이다.

두 프로쎄스들은 같은 이름으로 등록될수 없으므로 dcopserver는 충돌을 발견하고 등록이름을 수정한다. 첫째 충돌은 "wilbur-2"인 등록이름에서 발생하고 다음에는 "wilbur-3"에서 생긴다. 또한 registerAs()의 둘째 인수로서 TRUE를 사용하여 프로쎄스식별번호가 이름의 부분으로 추가되게 함으로써 유일한 등록이름을 생성하도록 한다. 례를 들면 실례wilbur의 프로쎄스ID가 34212이라면 등록이름은 "wilbur-34212"이다. 이것은 항상 유일한 등록이름을 생성하도록 한다.

제일웃준위창문부품은 17행과 18행에서 설정된다. 창문부품에 할당된 이름은 "wilreceiver"이

다. 그것이 하나이상의 DCOPObject를 하나의 프로쎄스에 포함하는데 쓰이고 그 매개는 통보문들을 받아들이는데 쓰일수 있으므로 매개에 이름을 제공하여야 한다.

20～22행은 DCOPClient의 등록이름을 현시한다.

GUI응용프로그람의 기본순환고리는 24행의 exec()호출에 의해 실행된다. 25행의 detach()호출은 dcopserver로부터 등록을 삭제하기 위한것이다. 이것은 프로쎄스가 실행을 멈출 때마다 등록이 자동적으로 삭제되므로 반드시 필요한것은 아니다.

28행의 WilReceiver구성자는 QMultiLineEdit창문을 읽기전용으로 설정하며 이것은 편집할수 없는 본문을 현시한다는것을 보여준다.

34행의 process()는 dcopserver로부터 통보문이 도착할 때마다 호출된다. 메쏘드에 4개 인수가 있다.

const QCString function	호출하려는 수속의 이름과 인수형들
const QByteArray &data	호출한 수속에 넘기려는 인수들
QCString &replyType	수속으로부터 돌아오는 값의 자료형
QByteArray &replyData	돌림값.

39행의 if명령문은 함수와 자료형이 허용되는가 확인한다. 많은 국부수속을 사용할수 있다. 어느것을 호출하는가 결정하기 위한 시험을 추가할 필요가 있다.

> 알아두기: 용어에서 원격수속호출과 조금 삭갈리기 쉽다. 원격프로쎄스는 함수나 메쏘드인 cubeRoot(double)라는 이름의 수속호출을 요구한다. 혹은 그것은 간단히 inline실행일수 있으며 심지어 완전히 다른 언어로 실현된다. 대면부가 같으면 결과는 얻어지고 실제프로쎄스의 세부는 중요하지 않다.

수속에 넘기려는 인수(들)은 QByteArray에 보관되여 도착하므로 실제값을 꺼내기 위하여 42행에서 창조된 QDataStream을 사용하는것이 필요하다. 이 실례에는 1개 인수만 있고 그것은 43행에서 inValue에 얻는다. 메쏘드 cubeRoot()는 outValue에 보관된 결과들과 함께 44행에서 호출된다. 돌림값은 45행에서 창조된 출력흐름을 사용하여 46행의 replyData에 보관된다. 돌림값의 자료형은 47행의 replyType에 보관된다. 돌림값이 TRUE이면 성공을 의미한다.

> 알아두기: 통보문에 응답하는데 필요된 코드가 좀 서툴어보이는데 이것은 자동생성용으로 설계되였기때문이다. 이 전체 과정은 콤파일러에 의해 자동생성된 process()메쏘드의 내용에 기초하는 프로젝트가 있으므로 가까운 앞날에 모두 간단화되여야 하며 이것은 MOC콤파일러가 신호와 처리부용 코드를 생성하는것과 아주 류사하다..

57행의 cubeRoot()메쏘드는 double값을 받아들이고 그 3제곱뿌리를 돌려준다. 또한 들어오는 수, 그리고 그 뿌리를 창문의 본문행으로서 현시한다. 이 메쏘드는 원격으로 호출되지만 그것은 보통 메쏘드이고 국부적으로 호출될수 있다.

TellWilbur

```
1 /* tellwilbur.cpp */
```

```
2 #include <kapplication.h>
3 #include <qcstring.h>
4 #include <dcopclient.h>
5
6 int main(int argc,char **argv)
7 {
8     KApplication app(argc,argv, "tellwilbur");
9
10    DCOPClient *client = app.dcopClient();
11    QCString dcopID = client->registerAs(app.name());
12
13    QByteArray params;
14    QDataStream stream(params,IO_WriteOnly);
15    stream << (double)999.0;
16    if(!client->send("wilbur","wilreceiver",
17                  "cubeRoot(double)",params)) {
18           qDebug("Well, that didn't work!");
19    }
20
21    client->detach();
22    return (0);
23 }
```

이 프로그람은 wilbur에게 통보문을 보내며 응답을 기다리지 않는다.

DCOP을 사용하여 통신하려면 dcopserver에 등록하여야 한다. 이것은 KApplication객체를 만들고 그것을 리용하여 국부DCOPClient의 주소를 얻은 다음 이 응용프로그람의 이름을 가지고 registerAs()를 호출한다는것을 의미한다.

인수들을 QByteArray에 넣어보내므로 14행에서 QDataStream객체를 창조하고 15행에서 거기에 double인수를 보관한다. 16행의 send()호출은 통보문을 보내지만 응답을 기다리지 않는다. 첫째 인수는 "wilbur"이다. 이것은 통보문을 받아들이려는 응용프로그람의 등록이름이다. 둘째 인수는 "wilreceiver"이다. 이것은 응용프로그람내부에 있는 DCOPObject의 이름이다. 호출하려는 수속은 "cubeRoot(double)"이라고 이름지어진다. 마지막 인수(params)는 수속에 넘기려는 인수값들을 포함한다.

> 알아두기: 이미 설명한것처럼 등록이름은 "wilbur-29003"과 같이 거기에 추가된 번호를 가질수 있다. 실제이름을 찾으려면 응용프로그람이 DCOPClient클라스의 registeredApplications()를 호출해야 한다. 이 메쏘드는 등록된 이름들을 모두 포함한

QCStringList객체를 돌려주며 응용프로그람은 여기서 필요한 이름(이름들)을 탐색할수 있다.

send()메쏘드는 응답을 기다리지 않으므로 돌림값은 없다. 21행의 detach()호출은 dcopserver로부터 등록을 삭제한다.

AskWilbur

```
1 /* askwilbur.cpp */
2 #include <kapplication.h>
3 #include <qcstring.h>
4 #include <dcopclient.h>
5
6 int main(int argc,char **argv)
7 {
8       KApplication app(argc,argv, "askwilbur");
9
10      DCOPClient *client = app.dcopClient();
11      QCString dcopID = client->registerAs(app.name());
12
13      QByteArray params;
14      QByteArray reply;
15      QCString replyType;
16      QDataStream stream(params,IO_WriteOnly);
17      stream << (double)888.0;
18      if(!client->call("wilbur","wilreceiver",
19                      "cubeRoot(double) ",params,
20                      replyType,reply)) {
21              qDebug("Well, that didn’t work!");
22      } else {
23              QDataStream inStream(reply,IO_ReadOnly);
24              if(replyType == "double") {
25                      double root;
26                      inStream >> root;
27                      QString str;
28                      str.sprintf("The return value is %g",root);
29                      qDebug(str);
30              }
```

```
31    }
32
33    client->detach();
34    return  (0);
35 }
```

이 실례는 이전 실례와 같은 일을 수행하지만 결과를 기다렸다가 현시한다.

18행의 call()호출은 통보문을 보내고 결과를 기다린다. 호출은 이전 실례와 같지만 20행에 2개의 돌림값이 있다. replyType인수는 돌림값의 자료형을 가지고 돌아오며 reply인수는 실제돌림값을 포함한다.

call()호출이 성공하면 23행에서 되돌아온 QByteArray로부터 값들을 읽어들이기 위하여 QDataStream을 창조한다. 돌림값의 자료형은 24행에서 검사되고 26행에서 국부변수에 보관되여 str라는 문자렬을 만드는데 사용되며 그다음 현시된다.

출력은 다음과 같다.

```
The return value is 9.61179
```

그림 15-1에서는 한 통보문을 tellwilbur로부터 받고 다른 통보문은 askwilbur로부터 받은 후에 wilbur에 의해 현시된 창문을 보여준다.
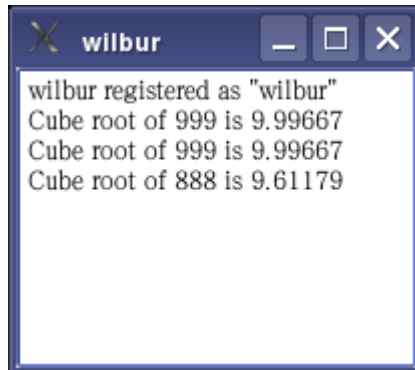
그림 15-1. 세 통보문을 받은 후의 Wilbur

제2절. 지령행인수

KCmdLineArgs클라스는 지령행인수를 확인하고 해석하는 대부분의 일을 조종할뿐아니라 모든 KDE응용프로그람들의 지령행인수들이 일관하도록 한다. 아래의 프로그람은 KCmdLineArgs를 사용하는 기초를 보여준다.

CommandLine

```
1 /* commandline.cpp */
2 #include <kcmdlineargs.h>
3 #include <iostream.h>
4
```

```
5 static KCmdLineOptions options[] = {
6     {"x","A binary option",0},
7     {"o <name>","An option with data","/dev/null"},
8     {"longbin", "A binary option",0},
9     {"longdata <name>","An option with data","/dev/null"},
10    {"t",0,0},
11    {"twoforms", "Two forms of a binary option",0},
12    {0,0,0}
13 };
14
15 int main(int argc,char **argv)
16 {
17    QCString option;
18    KCmdLineArgs::init(argc,argv,
19            "commandline",
20            "Example of command line parsing",
21            "Version 0.0");
22    KCmdLineArgs::addCmdLineOptions(options);
23    KCmdLineArgs *pargs = KCmdLineArgs::parsedArgs();
24
25    if(pargs->isSet("x"))
26            cout << " -x is set" << endl;
27    else
28            cout << " -x is not set" << endl;
29    option = pargs->getOption(“o”);
30    cout << " -o is set to " << option << endl;
31    if(pargs->isSet("longbin"))
32            cout << " --longbin is set" << endl;
33    else
34            cout << " --longbin is not set" << endl;
35    option = pargs->getOption("longdata");
36    cout << " --longdata is set to " << option << endl;
37
38    pargs->clear();
39    return (0);
```

40 }

사용할수 있는 지령행인수들은 5행의 KCmdLineOptions객체들의 배렬로서 정의된다. 배렬에서 선택한 지령은 3개 문자렬로 이루어져있다. 첫째 문자렬은 지령행에 나타나는 문자(렬)이고 둘째는 선택한 지령의 간단한 설명 그리고 셋째는 선택한 지령의 초기값문자렬이다. 선택들의 배렬은 12행에서 3개의 null문자렬을 포함하고있는 값들로 끝난다.

18행의 정적메쏘드 init()호출은 KCmdLineArgs클라스의 정적자료를 초기화한다. 처음 2개 인수는 C++지령으로부터 들어오는 표준argc와 argv변수이다. 이것들은 프로그람의 이름, 프로그람의 간단한 설명 그리고 프로그람의 현재판번호이다.

22행의 addCmdLineOptions()호출은 KCmdLineArgs클라스안에 KCmdLineOptions표를 보관한다. KCmdLineArgs클라스내부에 미리 정의된것들과 함께 이 선택목록은 가능한 모든 선택의 설정값을 결정하는데 필요한 정보이다.

23행의 정적메쏘드 parseArgs()호출은 정의된 선택들에 대하여 지령행을 해석한다. 오유가 없으면 이 메쏘드는 자기 프로그람이 받아들일 준비가 되여있는 인수값들을 가진 KCmdLineArgs객체의 지적자를 돌려준다. 지령행에서 모순되는 인수가 발견되면 프로그람은 오유통보문을 현시하고 프로그람을 정지한다.

6행에서 지정된 -x선택은 2진기발이다. 즉 그것이 지령행에 나타났는가 아닌가하는 정보를 가진다. 25행의 isSet()호출은 값이 지령행에 나타났으면 TRUE를, 그렇지 않으면 FALSE를 돌려준다.

7행에서 지정된 -o선택은 지령행에서 값이 그뒤에 올것을 요구하는 선택이다. 지령행에 값이 없을 때 사용하는 기정값문자렬을 제공한다. 29행의 getOption()호출은 인수값을 얻는다.

선택이름이 1문자이상이면 지령행에서 2개의 풀이표를 요구한다. 8행에서 정의한 --longbin선택은 31행에서 isSet()호출에 의해 검사되는 2진기발이다. --longdata선택은 자료가 그뒤에 올것을 요구하며 그 값은 35행의 getOption()호출에 의해 얻는다.

10행과 11행은 같은것을 의미하는 2개의 기발을 지정하는 한가지 실례이다. 둘째와 셋째 인수들을 모두 null지적자로 함으로써 -t선택이 –형식선택의 동의어로 된다. 지령행에서 그리고 프로그람내에서 그것을 사용할수 있다.

38행의 clear()호출은 프로그람이 완료하려고 하기때문에 이 실례에서는 필요하지 않지만 인수자료가 매우 큰 경우에 할당된 기억기를 해방하는데 이 메쏘드를 사용할수 있다.

이 실례에서 다음 지령행은 두가지 기발을 지정한다.

```
commandline -x --longdata /mnt/fred
```

프로그람에 의해 현시된 본문은 다음과 같다.

```
- x is set
-o is set to /dev/null
--longbin is not set
--longdata is set to /mnt/fred
```

오유가 있으면 23행의 parseArgs()호출은 프로그람을 중지하고 통보문을 현시한다. 실례로 다음 지령행은 알려지지 않은 기발을 지정한다.

```
commandline -x -j
```

출력은 프로그람이름을 포함하며 다음과 같은 알려지지 않은 선택을 지정한다.

```
commandline: Unknown option '-j'.
commandline: Use --help to get a list of available command line options.
```

–help선택을 리용하면 가능한 선택들의 전체목록을 얻을수 있다.

```
Usage : commandline [ Qt-options ] [ KDE-options ] [ options ]
Example of command line parsing
Generic options :
    --help              Show help about options
    --help-qt           Show Qt specific options
    --help-kde          Show KDE specific options
    --help-all          Show all options
    --author            Show author information
    -V, --version       Show version information
    --                  End of options
Options:
    -x                  A binary option
    -o <name>           An option with data [/dev/null]
    --longbin           A binary option
    --longdata <name>   An option with data [/dev/null]
    -t, --twoforms      Two forms of a binary option
```

제3절. 유일한 응용프로그람

어떤 응용프로그람들은 어떤 순간에 실행중에 있는 자기 사본이 2개이상 되지 않도록 보호하여야 한다. 이것은 응용프로그람을 DCOP봉사기로 등록함으로써 달성되며 그자체가 이미 등록되였으면 다른 사본이 벌써 실행되고있다고 가정한다. 다음 실례는 프로그람의 실례가 2개이상 있을수 없다는것을 담보하기 위하여 KApplication대신에 KUniqueApplication을 사용한다.

Unique

```
1 /* unique.cpp */
2
3 #include <kuniqueapp.h>
4 #include <kaboutdata.h>
```

```
5 #include <kcmdlineargs.h>
6 #include <qlabel.h>
7 #include <iostream.h>
8
9 static KCmdLineOptions options[] = {
10     {"x", "A Binary option",0},
11     {0,0,0}
12 };
13
14 int main(int argc,char **argv)
15 {
16     KAboutData about("unique",
17             "Example of unique application",
18             "0.1");
19     KCmdLineArgs::init(argc,argv,&about);
20     KCmdLineArgs::addCmdLineOptions(options);
21     KUniqueApplication::addCmdLineOptions();
22
23     if(!KUniqueApplication::start()) {
24             cout << "Unique is already running" << endl;
25             exit(0);
26     }
27
28     KUniqueApplication kuapp;
29     QLabel *label = new QLabel("Unique",0);
30     label->setAlignment(Qt::AlignVCenter
31             | Qt::AlignHCenter);
32     label->show();
33     kuapp.setMainWidget(label);
34     return (kuapp.exec());
35 }
```

19행에서 KCmdLineArgs의 init()메쏘드호출은 지령행인수들을 해석하고 보관한다. KAboutData객체는 기본적인 응용프로그람정의문자렬들 즉 프로그람이름, 간단한 설명이름 그리고 판번호들을 포함한다. 20행의 addCmdLineOptions()호출은 9행의 표에서 소개한 선택들을 정의하는데 사용되며 21행의 addCmdLineOptions()호출은 KUniqueApplication클라스에 지정

된 선택들을 포함한다.

23행의 start()호출은 프로그람의 이 실례를 실행하려고 하는가, 또는 그것이 유일하지 않기때문에 완료하려고 하는가 알아야 할 때 필요하다. start()호출이 이루어지지 않으면 프로그람의 한개 사본이 이미 실행되고있으며 이 프로그람은 28행에서 KUniqueApplication객체를 창조하려고 시도할 때 조용히 정지된다.

KUniqueApplication클라스는 KApplication을 기초클라스로 사용하므로 28행에서 창조된 kuapp객체는 그것이 KApplication객체인것처럼 취급될수 있다. QLabel창문부품이 만들어지고 29～33행에서 기본창문창문부품으로 설치되며 응용프로그람의 실행순환고리는 34행에서 호출된다.

제4절. 실례애플레트

애플레트는 하나의 작은 창문을 현시하는 프로그람이고 창문은 보통 현시기의 아래에 나타나는 KDE구획에 있다. 이 창문한계외에 애플레트는 다른 프로그람처럼 크고 복잡할수 있다.

다음 실례애플레트는 본문을 포함하는 구획창문을 현시하며 kmail응용프로그람을 기동함으로써 마우스단추에 응답한다. 이것은 아주 간단한 애플레트이다. 사용할 때 응용프로그람이 우연히 여러번 기동되지 않도록 보호하는것이 필요하며 몇가지 반결합을 사용자에게 제공하여 애플레트가 마우스에 응답하고 있다는것을 사용자가 알게 한다.

구획이 수직 또는 수평으로 현시하도록 배렬할수 있고 창문크기조절규칙이 두 방향에서 약간 다르기때문에 애플레트는 매개 방향의 크기를 구획에 알려주어야 한다.

MailApplet머리부파일

```
1 /* mailapplet.h */
2 #ifndef MAILAPPLET_H
3 #define MAILAPPLET_H
4
5 #include <qfontmetrics.h>
6 #include <kpanelapplet.h>
7
8 class MailApplet: public KPanelApplet
9 {
10    Q_OBJECT
11 public:
12    MailApplet(QWidget *parent=0,const char *name=0);
13    int widthForHeight(int height);
14    int heightForWidth(int width);
```

```
15     void about();
16     void help();
17     void preferences();
18 protected:
19     void paintEvent(QPaintEvent *e);
20     void mousePressEvent(QMouseEvent *e);
21 };
22
23 #endif
```

애플레트의 기초클라스는 KPanelApplet이다. KPanelApplet는 QWidget를 기초클라스중의 하나로 사용하므로 코드는 창문에 대한 직접호출을 가진다. 10행의 마크로 Q_OBJECT는 다른 KDE창문응용프로그람처럼 MOC콤파일러에 의해 사용되므로 처리부와 신호의 표준형식을 사용할수 있다. 메쏘드 widthForHeight()와 heightForWidth()는 기초클라스에서 가상메쏘드로 선언되므로 그것들은 애플레트에서 실현되여야 한다.

MailApplet

```
1 /* mailapplet.cpp */
2 #include <kapplication.h>
3 #include <kcmdlineargs.h>
4 #include <kmessagebox.h>
5 #include <kaboutdialog.h>
6 #include <qpainter.h>
7 #include <stdlib.h>
8 #include "mailapplet.h"
9
10 #define vText "VERT"
11 #define hText "HORIZ"
12
13 int main(int argc,char **argv)
14 {
15     KCmdLineArgs::init(argc,argv,
16             "mailapplet",
17             "Mail Applet Example",
18             "Version 0.0");
19     KApplication app;
20     MailApplet *applet = new MailApplet(0, "mailapplet");
```

```
21     app.setMainWidget(applet);
22     applet->init(argc,argv);
23     return  (app.exec());
24 }
25 MailApplet::MailApplet(QWidget *parent,const char *name)
26     : KPanelApplet(parent,name)
27 {
28     setActions(About | Help | Preferences);
29     setFont(QFont("Courier",16,QFont::Bold));
30 }
31 void MailApplet::about()
32 {
33     KAboutDialog *about = new KAboutDialog(0, "mailapplet");
34     about->exec();
35 }
36 void MailApplet::help()
37 {
38     KMessageBox::information(0,
39             "The MailApplet Help Dialog");
40 }
41 void MailApplet::preferences()
42 {
43     KMessageBox::information(0,
44             "The MailApplet Preferences Dialog");
45 }
46 int MailApplet::heightForWidth(int width)
47 {
48     QFontMetrics fm = fontMetrics();
49     return  (fm.height());
50 }
51 int MailApplet::widthForHeight(int height)
52 {
53     QFontMetrics fm = fontMetrics();
54     return  (fm.width(hText));
55 }
```

```
56 void MailApplet::paintEvent(QPaintEvent *e)
57 {
58     QPainter p(this);
59     QFontMetrics fm = fontMetrics();
60     if(orientation() == Vertical) {
61             int y = height() / 2;
62             y += (fm.ascent() - fm.descent()) / 2;
63             int x = (width() - fm.width(vText)) / 2;
64             p.drawText(x,y,vText);
65     } else {
66             int y = height() / 2;
67             y += (fm.ascent() - fm.descent()) / 2;
68             int x = (width() - fm.width(hText)) / 2;
69             p.drawText(x,y,hText);
70     }
71 }
72 void MailApplet::mousePressEvent(QMouseEvent *e)
73 {
74     system("kmail &");
75 }
```

애플레트는 다른 응용프로그람과 아주 비슷하다. 중요한 차이는 기본창문부품이 KPanelApplet클라스를 기초클라스로 사용하는것이다.(또한 QWidget를 기초클라스로 사용한다.)

13행에서 시작하는 애플레트의 기본함수는 KCmdLineArgs를 사용하여 지령행정보를 읽어들이고 서술본문정보를 초기화한다. 19행에서 KApplication객체는 KCmdLineArgs의 init()메쏘드에 의해 보관된 대역정보를 사용하므로 인수없이 만들어진다. 이 응용프로그람의 기본창문부품은 20행과 21행에서 만들어진다. 22행에서 MailApplet의 기초클라스 KPanelApplet의 init()메쏘드호출은 애플레트에 지령행인수들을 넘긴다.

25행의 MailApplet구성자는 KPanelApplet구성자에 부모창문부품과 애플레트이름을 넘긴다. 28행의 setActions()호출은 3개 선택차림표항목중 어느것을 애플레트차림표로 포함해야 하는가 지정한다. (차림표를 표시하기 위하여 애플레트를 이동시키는 띠우에서 마우스오른쪽단추를 사용한다.) 이 실례에서 3개의 선택차림표항목들이 모두 나타난다. 29행의 setFont()호출은 창문부품의 기정서체를 설정한다.

About항목이 28행의 setActions()에 의해 지정되였으므로 31행의 about()메쏘드는 사용자가 차림표로부터 "About"를 선택할 때마다 호출된다. 이 실례는 간단히 빈 About칸을 현시한

다. 마찬가지로 Help와 Preference가 setActions()호출에서 지정되였으므로 Help와 Preferences차림표항목들은 36행과 41행에서 help()와 preference()메쏘드들을 호출하게 한다.

구획이 수평방향으로 현시될 때 모든 애플레트는 고정높이를 가지지만 너비는 변할수 있다. 너비를 결정하기 위하여 51행에서 메쏘드 widthForHeight()를 호출한다. 애플레트가 크기를 결심하여야 하는 경우에 높이값이 주어지면 이 메쏘드는 너비를 계산하여 돌려준다. 이 실례에서 그림 15-2와 같이 너비는 본문의 수평범위이다.

그림 15-2. 수평방향의 구획을 가지는 애플레트

구획이 수직방향으로 현시될 때 모든 애플레트는 고정너비를 가지지만 매개는 자체의 높이를 지정해야 한다. 그래서 46행의 메쏘드 heightForWidth()를 호출한다. 그림 15-3과 같이 이 실례에서 높이는 현시된 본문의 높이이다.

그림 15-3. 수직방향의 구획을 가지는 애플레트

56행의 paintEvent()메쏘드는 창문부품을 그릴 때마다 호출된다. 60행의 orientation()메쏘드는 구획의 방향에 따라 Vertical 혹은 Horizontal을 돌려준다. 이 실례에서 방향을 설명하는 본문이 선택되고 본문의 위치는 애플레트창문의 중심에 본문이 나타나도록 계산된다.

이 실례는 72행의 mousePressEvent()메쏘드를 실현하고 kmail응용프로그람을 기동하여 마우스찰칵에 응답한다.

요 약

응용프로그람들사이통신은 2개이상의 실행프로그람들을 요구하는 매우 복잡한 체계에서 아주 중요하다. 또한 표준메쏘드사용은 다른 프로젝트들의 부분으로서 씌여진 응용프로그람과의 통신을 가능하게 한다.

이 장은 다음과 같은것을 설명하였다.

- KDE는 dcopserver라는 중간배경프로쎄스를 통하여 프로쎄스사이 통보문들을 송수신한다.
- 지령행인수들을 읽고 처리하는데서 KCmdLineArgs클라스를 사용하면 지령행해석프로

그람작성이 간단해지고 모든 KDE응용프로그람에서 인수형식을 표준화한다.

• KApplication대신에 KUniqueApplication을 사용하면 오직 한개 사본의 프로그람을 한번에 실행한다.

다음 장은 파일들의 읽기와 써넣기, 날자와 시간조작과 같은 일을 처리하는데 리용할수 있는 몇가지 일반편의클라스들을 서술한다.

제16장. 일반편의클라스

학습내용

문자렬클라스에 의한 문자렬조작

프로그람이 끝나는 시간을 통지하는 시계의 실행

현재시간의 표식자와 경과시간의 검사

날자와 달력계산

파일로부터 본문의 읽어들이기

파일에 본문의 써넣기

GUI대면부를 만드는데 사용하는 클라스들과 함께 다른 일에 편리한 몇가지 편의클라스들이 있다. 특히 문자렬들과 고속으로, 능률적으로 작업하는 능력은 매우 중요하다. 자료의 현시와 얻기와 관련한 많은 문자렬조작에서 문자렬취급은 편의클라스들이 없으면 매우 많은 시간을 소비할수 있다.

응용프로그람작성에서 제기되는 다른 문제는 날자와 시간계산을 할수 있는 능력이다. 항상 여러가지 형식으로 시간을 돌려주는 조작체계호출이 있으며 시간값에 대한 복잡한 조작을 수행하려면 많은 프로그람을 작성해야 한다. 실례로 2개의 날자가 있다면 날자차이가 몇일인가를 어떻게 결정하겠는가?

대부분의 큰 자료파일들은 저장 및 복귀를 위해 자료기지패키지에 주어지지만 임의의 크기의 대다수 프로그람들은 구체적인 자료를 보관하는데 작은 본문파일들을 사용한다. C와 C++표준언어들이 파일들을 읽고 쓰는 아주 간단한 방법을 제공하지만 자료들을 포함하는 형식화와 비형식화문제가 여전히 존재한다.

이 장은 이 문제들을 해결하는 아주 편리한 클라스들을 설명한다. 그것은 Qt에서 사용할수 있는 모든 클라스의 완전목록을 의미하지 않으며 핵심클라스들중 일부를 포함한다.

제1절. 문자렬클라스

많은 프로그람작성은 문자렬조작을 포함한다. 특히 사용자대면부용프로그람을 만드는 경우에는 더 많이 리용된다. 자료는 현시하려는 문자렬들로 변환되고 사용자가 입력한 자료는 문자렬로부터 내부자료형식으로 바뀐다. 이 모두를 쉽게 조종하는 특수한 문자렬취급클라스가 있다.

1. QString

QString클라스는 가장 기초적인 문자렬클라스이다. QString클라스는 문자렬조작에 쓰이는 많은 메쏘드들을 가지고 있고 내부적으로 자료를 유니코드로 기억한다.

유니코드가 대단히 많은 문자들을 포함하는것을 제외하고 유니코드와 ASCII문자모임사이에 차이가 없다. 표준7bit ASCII 문자모임은 127개 문자로 제한되며 라틴자모, 아라비아수자, 구두점 그리고 일련의 조종문자(Carriage Return과 Escape 등) 등이 있다. 유니코드표준은

16bit문자를 사용하므로 65 536개 문자까지 포함할수 있다. 그러나 유니코드문자모임의 첫 127개문자(0～127의 수값)는 ASCII문자모임과 같으므로 ASCII를 유니코드로 바꿀 필요가 없다. 또한 라틴문자유니코드를 ASCII로 바꿀 필요도 없다.

참고: 유니코드사용법에 대한 더 많은 정보를 알려면 17장을 보시오.

다음 실례는 문자렬의 부분을 탐색하고 추출하는데 사용할수 있는 메쏘드들중 일부를 보여준다.

```
1 /* stringexamine.cpp */
2 #include <qstring.h>
3 #include <iostream.h>
4
5 int main(int argc,char **argv)
6 {
7     QString qstring;
8     QChar qchar;
9
10    qstring =
11        "There is much more to KDE than just a pretty face.";
12
13    cout << qstring << endl;
14    cout << "The string contains "
15        << qstring.length() << " characters." << endl;
16    qchar = qstring[4];
17    cout << "The 5th charater is '"
18        << (char)qchar.unicode() << "'." << endl;
19    cout << "The first 'u' is at "
20        << qstring.find('u') << "." << endl;
21    cout << "The last 'u' is at “
22        << qstring.findRev('u') << "." << endl;
23    cout << "The first 're' is at "
24        << qstring.find("re") << "." << endl;
25    cout << "The last 're' is at “
26        << qstring.findRev("re") << "." << endl;
27    cout << "There are "
28        << qstring.contains(‘e’) << " 'e's." << endl;
29    cout << "There are "
```

```
30            << qstring.contains("re") << " 're's." << endl;
31    cout << "The leading 7 characters are '"
32            << qstring.left(7) << "'." << endl;
33    cout << "The trailing 7 characters are '"
34            << qstring.right(7) << "'." << endl;
35    cout << "The 8 characters at index 22 are '"
36            << qstring.mid(22,8) << "'." << endl;
37
38    return  0;
39 }
```

이 프로그람의 출력은 다음과 같다.

```
There is much more to KDE than just a pretty face.
The string contains 50 characters.
The 5th charater is 'e'.
The first 'u' is at 10.
The last 'u' is at 32.
The first 're' is at 3.
The last 're' is at 39.
There are 5 'e's.
There are 3 're's.
The leading 7 characters are 'There i'.
The trailing 7 characters are 'y face. '.
The 8 characters at index 22 are 'KDE than'.
```

QString을 만드는데 쓰이는 여러가지 구성자들이 있다. QString은 간단한 문자배렬, QByteArray, QChar, QChar객체배렬, 다른 QString으로부터 만들수 있으며 또는 아무것도 지정하지 않고(0길이문자렬) 만들수 있다. QChar객체는 하나의 유니코드문자를 보관하는 용기인데 다음 장에서 설명한다.

문자렬조작을 제공하는 다중정의된 연산자들이 여러개 있다. 이 실례의 10행과 11행에서 대입연산자는 문자렬을 유니코드로 바꾸어 QString객체에 보관하는데 쓰인다. 또한 QString, QCString, QChar 그리고 char용의 대입연산자다중정의가 있다. 마찬가지로 +=연산자는 QString, QChar, 또는 char를 현존QString의 끝에 추가하는데 쓰인다.

20행과 24행에서 find()메쏘드들은 문자렬의 선두로부터 조사하여 문자 혹은 문자렬의 첫 출현을 발견하고 탐색한 부분문자렬의 선두문자의 첨수를 돌려준다. 22행과 26행의 findRev()메쏘드는 문자렬의 끝으로부터 조사하여 마지막 출현을 발견하고 부분문자렬의 선두의 첨수를 돌려준다. 28행과 30행의 contains()메쏘드들은 전체 문자렬을 조사하고 문자 또

는 부분문자렬의 출현회수를 돌려준다.

32행과 34행의 메쏘드 left()와 right()는 문자렬의 선두 혹은 끝에서 지정된 개수의 문자들을 포함하는 QString을 돌려준다. 36행의 mid()메쏘드는 문자렬의 첨수위치로부터 지정된 개수의 문자들을 포함하는 QString을 돌려준다. (이 실례에서 첨수는 22이고 문자수는 8이다.)

2. QString

QString의 내용을 변경하는데 많은 메쏘드를 사용할수 있다. 다음 실례는 일부 필요한 메쏘드들을 보여준다.

```
1 /* stringmodify.cpp */
2 #include <qstring.h>
3 #include <iostream.h>
4
5 QString init(QString str)
6 {
7       str = "There is more to KDE than a pretty face.";
8       return  str;
9 }
10
11 int main(int argc,char **argv)
12 {
13      QString qstring;
14
15      cout << "Unchanged: "
16              << init(qstring) << endl;
17      cout << "Uppper case: "
18              << init(qstring).upper() << endl;
19      cout << "Lower case: "
20              << init(qstring).lower() << endl;
21      cout << "Insert 'X': "
22              << init(qstring).insert(10,’X’) << endl;
23      cout << "Insert 'ABC': "
24              << init(qstring).insert(10,  "ABC") << endl;
25      cout << "Prepend 'X': "
26              << init(qstring).prepend(‘X’) << endl;
27      cout << "Prepend 'ABC': "
28              << init(qstring).prepend("ABC") << endl;
```

```
29    cout << "Append 'X': "
30            << init(qstring).append('X') << endl;
31    cout << "Append ‘ABC’: “
32            << init(qstring).append("ABC") << endl;
33    cout << "Remove 10: "
34            << init(qstring).remove(15,10) << endl;
35    cout << "Replace 10: "
36            << init(qstring).replace(15,10,  "ABC") << endl;
37
38    return 0;
39 }
```

출력은 다음과 같다.

```
Unchanged: There is more to KDE than a pretty face.
Uppper case: THERE IS MORE TO KDE THAN A PRETTY FACE.
Lower case: there is more to kde than a pretty face.
Insert 'X': There is mXore to KDE than a pretty face.
Insert 'ABC': There is mABCore to KDE than a pretty face.
Prepend 'X': XThere is more to KDE than a pretty face.
Prepend 'ABC': ABCThere is more to KDE than a pretty face.
Append 'X': There is more to KDE than a pretty face.X
Append 'ABC': There is more to KDE than a pretty face.ABC
Remove 10: There is more t a pretty face.
Replace 10: There is more tABC a pretty face.
```

이 실례는 QString의 매개 메쏘드가 QString객체의 내용을 변경하기때문에 5행에서 init()함수에 의하여 문자렬을 초기화한다.

18행과 20행의 upper()와 lower()메쏘드들은 문자렬안의 모든 영문자를 대문자 혹은 소문자로 변환한다.

22행과 24행의 insert()는 삽입하려는 문자들의 수만큼 문자렬의 일부를 오른쪽으로 이동하여 문자렬을 확장한다. 그다음 문자렬에 생겨난 짬에 인수로 넘긴 문자(들)이 삽입된다.

26행과 28행에서 prepend()메쏘드는 삽입하려는 문자수만큼 모든 문자렬을 오른쪽으로 이동하여 문자렬을 확장한다. 그 다음에 문자렬의 앞에 생긴 짬에 인수로 넘긴 문자(들)이 삽입된다.

30행과 32행에서 append()메쏘드는 삽입하려는 문자수만큼 문자렬을 확장하고 문자렬의 오른쪽끝에 생긴 짬에 문자들을 보관한다.

34행의 remove()메쏘드는 지정된 문자수만큼 나머지 문자렬을 오른쪽으로 옮기여 문자

렬을 축소한다. 이리하여 중심의 문자모임을 문자렬로부터 효과적으로 삭제하면서 그 우에 다시 써넣는다. 이 실례에서 첨수는 15이고 제거된 문자수는 10이다.

36행의 replace()메쏘드는 문자렬을 확장 및 축소하거나 같은 길이로 만드는데 쓰인다. 문자렬안의 일부 문자들은 교체된다. 그 과정은 remove()다음에 insert()한것과 기능적으로 같다. 첨수위치(이 실례에서 15)의 오른쪽 문자들은 삽입 혹은 삭제하려는 문자수에 따라 오른쪽으로 또는 왼쪽으로 옮겨진다(이 실례에서 10문자를 삭제하고 3개를 삽입해야 하므로 7개 문자만큼 왼쪽으로 옮긴다). 지정된 문자렬은 그다음 첨수위치에서 다시 써넣는다.

3. QString-

수값과 수값의 문자표시사이의 직접변환메쏘드들이 QString클라스의 부분으로서 포함되여있다. 다음 실례는 그 동작을 보여준다.

```
1 /* stringnumber.cpp */
2 #include <qstring.h>
3 #include <iostream.h>
4
5 int main(int argc,char **argv)
6 {
7     QString qstring;
8     bool Ok;
9
10    int inum = 9421;
11    qstring.setNum(inum);
12    cout << "Short string: " << qstring << endl;
13    inum = qstring.toInt(&Ok);
14    cout << "Short value: " << inum << endl;
15
16    double dnum = 2813.8282190;
17    qstring.setNum(dnum);
18    cout << "Double string at 6: " << qstring << endl;
19    dnum = qstring.toDouble(&Ok);
20    cout << "Double value at 6: " << dnum << endl;
21
22    dnum = 2813.8282190;
23    qstring.setNum(dnum, 'g',9);
24    cout << "Double string at 9: " << qstring << endl;
25    dnum = qstring.toDouble(&Ok);
```

```
26     cout << "Double value at 9: " << dnum << endl;
27
28     ulong ulnum = 0xCFA90B2;
29     qstring.setNum(ulnum,16);
30     cout << "Ulong string: " << qstring << endl;
31     ulnum = qstring.toULong(&Ok,16);
32     cout << "Ulong value: " << ulnum << endl;
33
34     qstring.sprintf("The int is %d and the long is %lu",
35            inum,ulnum);
36     cout << qstring << endl;
37
38     return 0;
39 }
```

프로그람의 출력은 다음과 같다.

```
Short string: 9421
Short value: 9421
Double string at 6: 2813.83
Double value at 6: 2813.83
Double string at 9: 2813.82822
Double value at 9: 2813.83
Ulong string: cfa90b2
Ulong value: 217747634
The int is 9421 and the long is 217747634
```

이 실례는 몇가지 자료형들을 보여준다. 여기서 보여준 int, double, ulong자료형들과 함께 long , uint , short , ushort , char 및 QChar에서도 같은 수법을 사용할수 있다.

11행의 setNum()호출은 12행에서 cout에 의해 보여준것처럼 int값을 문자렬로 변환한다. 13행의 toInt()호출은 QString의 문자들을 읽어들이고 14행에서 보여준것처럼 int값으로 변환한다. 변환이 성공하면 toInt()에 넘긴 론리인수는 TRUE로 되고 그렇지 않으면(실례로 문자렬에 수자가 없으면) FALSE로 된다.

17행의 setNum()호출은 double값을 문자렬로 변환한다. 기정은 18행에서 cout문에 의해 보여준것처럼 6자리를 포함하는 수자이다. 수자수를 줄일 때 값을 간단히 잘라버리지 않고 반올림한다. 19행에서 toDouble()호출에 의해 문자렬을 수값형식으로 역변환하였을 때 그것은 오직 문자렬에 보관되였던 6개 수자를 포함한다.

22～26행은 이전처럼 같은 double값을 사용하지만 이번에는 수자들의 개수는 9로 설정

되고 문자렬이 더 길어지고 좀 더 정확하다. 문자 g는 문자렬을 형식화하는데 사용된다. 이것은 기정이지만 표준 sprintf()실수선택 즉 f, F, e, E, g 및 G를 사용할수 있다.

29행의 setNum()호출에서 보여주는것처럼 10이 아닌 기수로 값을 변환할수 있다. 이 실례는 ulong형의 32bit 16진수를 문자렬로 변환한다. 31행의 toULong()호출은 16진문자들의 문자렬을 2진수형식으로 변환한다. setNum()메쏘드는 수를 전부 소문자로 바꾸지만 toULong()은 소문자와 대문자를 모두 변환한다.

34행에 보여준것처럼 sprintf()를 사용할수 있다. 인수형식은 표준 C와 같고 결과문자렬은 QString객체내부에 보관된다.

4. QString

내부변환기는 대화칸의 본문이나 단추나 표식자의 본문으로 현시하려는 문자렬들을 구성하는데 아주 편리하다. 그것은 sprintf()함수와 류사하지만 자동적으로 다른 자료형들을 탐색하므로 사용하기 더 쉽다. 다음 실례는 그것이 2개의 값들과 작업하는 방법을 보여준다.

```
1 /* stringargs.cpp */
2 #include <kapplication.h>
3 #include <qlabel.h>
4
5 class StringArgs: public QWidget
6 {
7 public:
8     StringArgs(QWidget *parent=0,const char *name=0);
9 };
10 int main(int argc,char **argv)
11 {
12    KApplication app(argc,argv,  "stringargs");
13    StringArgs stringargs;
14    app.setMainWidget(&stringargs);
15    stringargs.show();
16    return  app.exec();
17 }
18
19 StringArgs::StringArgs(QWidget *parent,const char *name)
20    : QWidget(parent,name)
21 {
22    int y = 60;
23    int x = 210;
```

```
24    QString str;
25
26    resize(x,y);
27    str = tr("Width is %1 and height is %2").arg(x).arg(y);
28    QLabel *label = new QLabel(str,this);
29    label->setGeometry(20,20,170,20);
30 }
```

27행의 tr()메쏘드는 QString객체를 만들고 2개 arg()메쏘드에 의하여 자료를 문자렬들로 변환하고 %1와 %2로 표식된 위치에 그것들을 삽입한다. 모든 자료를 변환하여야 할 때 많은 arg()메쏘드들을 사용할수 있다. arg()메쏘드들은 위치관계를 가지며 첫째는 %1에 대응하고 둘째는 %2에 대응된다. 이 프로그람에 의해 현시된 창문은 그림 16-1에 보여준다.
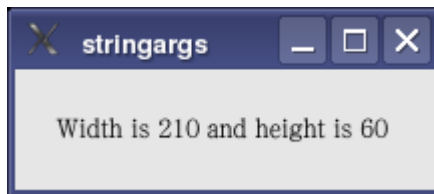
그림 16-1. 자료변환과 형식화

이 실례의 int자료형외에 long , ulong , uint , short , ushort , char , QChar , QString 및 double용 arg()메쏘드들이 다중정의된다. 변환메쏘드가 QObject::tr()이므로 이러한 변환은 오직 QObject를 계승하는 객체들에서만 사용될수 있다.

5. QString

문자렬에서 공백문자에는 타브(\t), 행바꾸기(\n), 종이바꿈(\f), 복귀(\r) 및 공백기호들이 속한다. 문자렬에서 공백을 제거하는데 편리한 메쏘드가 있다. 다음것은 실례문자렬이다.

```
"     This has\t tabs    \nand\nnewlines in it        \n        "
```

공백은 아래와 같은것을 리용하여 제거할수 있다.

```
str.stripWhiteSpace();
```

결과는 다음과 같다.

```
"This has\t tabs \nand\nnewlines in it"
```

행의 앞과 뒤 또한 안으로부터 공백을 다음과 같은 방법으로 삭제할수 있다.

```
str.simplifyWhiteSpace();
```

앞과 뒤는 잘리우고 다수의 내부공백문자는 한개 공백으로 줄어든다. 결과는 다음과 같다.

```
"This has tabs and newlines in it"
```

6. QStringList.

QStringList객체는 QString객체의 가변길이배렬이다. 이것을 매우 효과적으로 사용하여 특수한 조작을 수행할수 있다. 다음 실례는 배렬에 문자렬들을 삽입하는 방법과 문자렬들을 조작하는 방법들을 보여준다.

```
1 /* stringlist.cpp */
2 #include <qstringlist.h>
3 #include <iostream.h>
4
5 int main(int argc,char **argv)
6 {
7     QStringList list;
8
9     list.append("First");
10    list += "Second";
11    list << "Third" << "Fourth" << "Fifth";
12    for(int i=0; i<list.count(); i++)
13        cout << list[i] << endl;
14
15    QString joined = list.join("^");
16    cout << endl << joined << endl;
17
18    list << "Apple" << "apple";
19    list.sort();
20    cout << endl;
21    for(int i=0; i<list.count(); i++)
22        cout << list[i] << endl;
23
24    list = list.grep("e");
25    cout << endl;
26    for(int i=0; i<list.count(); i++)
27        cout << list[i] << endl;
28
29    list = QStringList::split("|",  "Make|a|list|from|this");
30    cout << endl;
31    for(int i=0; i<list.count(); i++)
32        cout << list[i] << endl;
33
34    list[1] = "This replaces the 'a' string";
35    cout << endl;
```

```
36    for(int i=0; i<list.count(); i++)
37          cout << list[i] << endl;
38
39    return  0;
40 }
```

이 프로그램의 출력은 다음과 같다.

```
First
Second
Third
Fourth
Fifth
First^Second^Third^Fourth^Fifth
Apple
Fifth
First
Fourth
Second
Third
apple
Apple
Second
apple
Make
a
list
from
this
Make
This replaces the ‘a’ string
list
from
this
```

9～11행은 문자렬을 배렬의 끝에 추가하는 각이한 방법들을 보여준다. append()메쏘드와 +=연산자는 둘다 배렬끝에 문자렬을 추가한다. <<연산자도 같은 조작을 하며 하나의 조작으로 여러개 문자렬을 추가할수 있다. 문자렬들이 코드에 추가되는 순서로 보관된다는것을 출

력으로부터 알수 있다. []연산자는 문자렬들을 첨수값에 따라서 얻을수 있도록 다중정의된다.

15행의 join()메쏘드는 문자렬들사이에 특정한 분리기호문자렬을 삽입함으로써 배렬의 모든 성원들로부터 하나의 긴 문자렬을 만든다.

18행에서 2개 성원을 배렬끝에 추가한다. 19행의 sort()메쏘드는 문자들의 수값을 사용하여 문자렬들의 배렬을 정렬한다. 이것은 매우 빠른 정렬이고 많은 실례에서 사용할수 있지만 요구하지 않는 순서로 정렬하는 때가 있다. 실례로 출력결과에서 보여준것처럼 대문자는 항상 소문자앞에 온다.

24행의 grep()호출은 메쏘드호출에서 지정된 표현과 일치하는 문자들만 포함하는 새로운 QStringList객체를 만든다. 이 실례에서는 문자 e를 포함하는 문자렬들만 새로운 QStringList객체에 보관한다.

QStringList객체는 특정한 구분기호문자렬을 사용하여 단일문자렬을 분리할수 있다. 2개의 련이은 구분기호들을 하나로 고찰해야 한다면 TRUE로 설정하고 또는 길이가 0인 문자렬이 허용되면 FALSE로 설정하는 3번째 인수가 있다(실례에서 사용하지 않았다). 기정은 TRUE이다.

34행의 []연산자는 현재 문자렬을 그 첨수를 사용하여 교체할수 있다. 즉 첨수값이 유효하면 배렬크기를 이러한 방법으로 조절할수 있다.

제2절. 시계의 실행

13장에서 bouncer프로그람은 QTimer객체를 사용하여 한번실행내부시계를 실현하는 실례를 포함하지만 신호대신에 사건을 사용하여 련속시계를 실행하는 더 간단한 방법도 있다. 다음 실례는 QObject클라스의 메쏘드들을 사용함으로써 련속시계들을 실현하는 방법을 보여준다.

```
1 /* stringlist.cpp */
2 #include <qapplication.h>
3 #include <iostream.h>
4
5 class TwoTimer: public QObject
6 {
7 public:
8     TwoTimer(QObject *parent=0,const char *name=0);
9 private:
10    int ID1;
11    int ID2;
12    bool timer2;
13 protected:
```

```
14    void timerEvent(QTimerEvent *event);
15 };
16
17 TwoTimer::TwoTimer(QObject *parent,const char *name)
18    : QObject(parent,name)
19 {
20    ID1 = startTimer(2000);
21 }
22 void TwoTimer::timerEvent(QTimerEvent *event)
23 {
24    if(event->timerId() == ID1) {
25            cout << "Timer 1" << endl;
26            if(timer2) {
27                    killTimer(ID2);
28                    timer2 = FALSE;
29            } else {
30                    ID2 = startTimer(200);
31                    timer2 = TRUE;
32            }
33    } else if(event->timerId() == ID2) {
34            cout << "Timer 2" << endl;
35    }
36 }
37
38 int main(int argc,char **argv)
39 {
40    QApplication app(argc,argv);
41    TwoTimer *to = new TwoTimer();
42    return app.exec();
43 }
```

필요할 때 많은 시계들을 동시에 실행할수 있다. 매 시계에는 실행시에 유일식별번호가 할당된다. 이 실례에서 TwoTimer클라스는 2개 시계를 실행하며 10행과 11행에서 ID1과 ID2에 식별번호들을 보관한다.

17행에서 시작하는 구성자는 20행에서 startTimer()를 호출하여 시계들중 하나를 실행시킨다. 이 시계는 2000ms(2s)마다 실행되며 ID1로서 보관된 식별번호를 가지고있다.

메쏘드 timerEvent()는 QObject클라스에서 정의된 보호메쏘드이고 이 실례의 22행에서 재정의된다. 모든 시계는 이와 같은 사건메쏘드를 호출한다. QTimerEvent객체는 시계의 식별번호를 포함하므로 어느 시계가 끝나고 메쏘드호출을 일으켰는지 결정하는것은 상대적으로 간단하다.

이 실례에서 첫째 시계의 작용이 끝날 때마다 둘째 시계가 기동하거나 정지한다. 26행에서 론리형의 timer2가 검사된다. 그것이 TRUE이면 둘째 시계는 실행되고있고 그것은 killTimer()를 호출함으로써 정지된다. 그것이 FALSE이면 둘째 시계는 200ms(2/10s)의 시간간격으로 startTimer()을 호출하기 시작한다.

38행에서 시작하는 프로그람의 기본함수는 QApplication객체를 만들고 시계들을 조종하는 Qt체계를 초기화한다. 시계를 만든 후에 QApplication객체의 exec()가 호출된다. 시간조정은 모두 exec()내부에서 관리된다.

시계들을 취급하는 다른 메쏘드가 있으나 그것은 이 실례에서 사용되지 않는다. 메쏘드 killTimers()호출은 모든 시계들을 정지한다.

제3절. QDate클라스

다음 실례는 1752년부터 약 8000년까지 모든 날자를 포함할수 있는 QDate클라스들을 설명한다. 1752가 아래한계인 리유는 그것이 윤년을 가지는 그레고리달력의 시작을 표식하는 해이기때문이다. (그해에 달력이 어떻게 조절되는가 알기 위하여 지령행으로부터 cal 1752이라고 입력하여 9월을 고찰한다.) 또한 1년은 2자리값(00～99)으로 QDate객체에 입력할수 있고 세기 1900이 가정된다.

```
1 /* showdate.cpp */
2 #include <qdatetime.h>
3 #include <qstring.h>
4 #include <iostream.h>
5
6 int main(int argc,char **argv)
7 {
8      if(argc != 4) {
9              cout << "Usage: showdate <yy> <mm> <dd>" << endl;
10             return  1;
11     }
12     QString yy = argv[1];
13     QString mm = argv[2];
14     QString dd = argv[3];
15     QDate date(yy.toInt(),mm.toInt(),dd.toInt());
```

```
16    if(!date.isValid()) {
17            cout << "Invalid  date" << endl;
18            return(2);
19    }
20
21    cout << "Date: " << date.toString() << endl;
22    cout << "yyyy/mm/dd: " << date.year() << "/"
23            << date.month() << "/" << date.day() << endl;
24    cout << "Day of week: " << date.dayOfWeek() << " ("
25            << date.dayName(date.dayOfWeek()) << ")" << endl;
26    cout << "Month name: "
27            << date.monthName(date.month()) << endl;
28    cout << "Day of year: " << date.dayOfYear() << endl;
29    cout << "Days in month: "
30            << date.daysInMonth() << endl;
31    cout << "Days in year: " << date.daysInYear() << endl;
32
33    return 0;
34 }
```

이 프로그람을 실행하기 위하여 년, 월, 일을 지령행에 입력한다.

```
showdate 1964 3 12
```

프로그람의 출력은 다음과 같다.

```
Date: Thu Mar 12 1964
yyyy/mm/dd: 1964/3/12
Day of week: 4 (Thu)
Month name: Mar
Day of year: 72
Days in month: 31
Days in year: 366
```

QDate에 날자를 보관하는 2가지 방법이 있다. 한가지 방법은 이 실례와 같이 구성자에서 년, 월, 일값들을 지정하는것이다. 다른 방법은 setYMD()호출에서 년, 월, 일을 지정하는것이다. QDate객체로부터 정보를 사용하기전에 날자가 유효한가 담보해야 한다. 이 실례의 16행에 보여준것처럼 isValid()호출은 꼭 필요하다. 또한 setYMD()메쏘드는 날자가 유효한가 가리키는 bool값을 돌려준다.

현재날자를 포함한 QDate객체를 정적메쏘드에 의해 만들수 있다.

```
QDate qdate = QDate::currentDate();
```

또한 앞뒤로 옮길 날자수를 지정하여 날자를 조종할수 있다. 실례로 15일 앞으로 날자를 옮길 때 다음 행을 사용한다.

```
QDate date2 = date.addDays(15);
```

날자수가 부수이면 새로운 QDate객체는 15일 이전의 날자를 가진다. 어떠한 지정된 점까지 날자를 조종하는데 다른 유효값을 사용할수 있다. 실례로 다음은 다음 달의 첫날까지 날자를 이동한다.

```
QDate date2 = date.addDays(date.daysInMonth()-date.day()+1);
```

한 날자로부터 다른 날자까지의 정이거나 부인 날자수는 다음과 같이 결정할수 있다.

```
int days = date.daysTo(date2);
```

끝으로 다음과 같은 비교연산자들은 두 날자사이의 관계를 결정할수 있다.

```
if(date1 == date2) ...
if(date1 != date2) ...
if(date1 < date2) ...
if(date1 > date2) ...
if(date1 <= date2) ...
if(date1 >= date2) ...
```

제4절. QTime클라스

QTime클라스는 계산이 규칙적이라는데서 QDate클라스보다 더 간단하다(월과 년과는 달리 시와 분은 같은 길이를 가지고있다). 그러나 그것을 경과시계로 사용할수 있다는 사실은 QTime클라스에 복잡성을 더해준다. 여기에 실례가 있다.

```
1 /* showtime.cpp */
2 #include <qdatetime.h>
3 #include <iostream.h>
4 #include <unistd.h>
5
6 int main(int argc,char **argv)
7 {
8     if(argc != 5) {
9             cout << "Usage: showtime <hh> <mm> <ss> <ms>"
10                    << endl;
11            return  1;
12    }
13    QString hh = argv[1];
```

```
14     QString mm = argv[2];
15     QString ss = argv[3];
16     QString ms = argv[4];
17     QTime qtime(hh.toInt(),mm.toInt(),
18     ss.toInt(),ms.toInt());
19     if(!qtime.isValid()) {
20             cout << "Invalid time" << endl;
21             return  2;
22     }
23
24     cout << "Time: " << qtime.toString() << endl;
25     cout << "hh:mm:ss.ms: " << qtime.hour() << ":"
26             << qtime.minute() << ":" << qtime.second() << "."
27             << qtime.msec() << endl;
28
29     qtime.start();
30     cout << "Start time: " << qtime.toString() << endl;
31     sleep(2);
32     int milliseconds = qtime.restart();
33     cout << "Restart time: " << qtime.toString()
34             << " (after " << milliseconds << " milliseconds)"
35             << endl;
36     for(int i=0; i<5; i++) {
37             cout << "Elapsed: " << qtime.elapsed() << endl;
38             sleep(1);
39     }
40
41     return 0;
42 }
```

이 프로그램의 출력은 다음과 같다.

```
Time: 15:04:22
hh:mm:ss.ms: 15:4:22.431
Start time: 08:42:07
Restart time: 08:42:09 (after 2010 milliseconds)
Elapsed: 1
```

```
Elapsed: 1010
Elapsed: 2020
Elapsed: 3030
Elapsed: 4040
```

QTime구성자는 ms(0～999)로 된 초의 소수부인 4번째 인수를 가지고있다. 4번째 인수는 선택적이고 기정값은 0이다. 시, 분, 초에 대하여 범위밖의 수를 입력하는것을 피하기 위하여 QTime객체를 사용하기전에 시간이 정확한가를 isValid()를 호출하여 확인해야 한다.

29행의 start()메쏘드는 2가지 일을 한다. 즉 현재시간을 가지고 QTime객체를 적재하고 경과시계값을 0으로 설정한다. 또한 정적메쏘드 currentTime()을 포함하는 새로운 QTime객체를 만들수 있다.

```
QTime qtime = QTime::currentTime();
```

32행의 restart()호출은 또한 현재시간으로 QTime객체를 설정하고 경과시계를 기동하고 경과시계의 값을 돌려주는데 이것은 start() 혹은 restart()의 마지막 호출후에 ms의 값이다. 출력으로부터 알수 있는것처럼 경과시간은 2s(2000ms)에다가 값을 현시하는데 걸리는 추가시간인 10ms값을 더한 값이다.

36행에서 시작하는 순환은 elapsed()를 반복 호출하고 1s동안 정지한다. 출력으로부터 경과시간이 증가하고있고 경과시계를 읽고 현시하는데 10ms걸린다는것을 알수 있다.

알아두기: 경과시계는 24h동안만 동작하고 그때 자동적으로 0으로 재설정된다. 또한 QTime의 현재시간은 콤퓨터의 체계시계에 기초하므로 다른 방법으로 체계시계를 변경하면 경과시계에 영향을 미친다.

2가지 메쏘드를 사용하여 미래 또는 과거로 시간을 조절할수 있다. 그것들은 새로운 시간값을 포함하는 새로운 QTime객체를 만들고 호출한다.

```
QTime qtime2 = qtime.addSecs(int seconds);
QTime qtime2 = qtime.addMSecs(int milliseconds);
```

부의 값은 시간을 과거로 돌리고 정의 값은 미래로 넘긴다. 시간에 충분한 값을 더하거나 덜어서 자정을 넘길수 있다. 실례로 현재시간이 23:59:45일 때 30s를 더하면 결과는 00:00:15이다.

다음의 비교연산자들은 두 시간들사이의 관계를 결정한다.

```
if(time1 == time2) ...
if(time1 != time2) ...
if(time1 < time2) ...
if(time1 > time2) ...
if(time1 <= time2) ...
if(time1 >= time2) ...
```

제5절. QDateTime클라스

QDateTime클라스는 근본적으로 QDate클라스와 QTime클라스를 결합한것으로서 두 클라스를 하나로 취급하는 메쏘드들을 가지고있다. 다음 실례는 QDateTime클라스에서 쓰이는 기본조작을 보여준다.

```
1 /* showtime.cpp */
2 #include <qdatetime.h>
3 #include <iostream.h>
4 #include <time.h>
5
6 int main(int argc,char **argv)
7 {
8     time_t bintime;
9
10    QDate qdate(2002,5,12);
11    QTime qtime(04,32,58);
12    QDateTime dt(qdate,qtime);
13    if(!dt.isValid()) {
14        cout << "Invalid QDateTime" << endl;
15        return  2;
16    }
17
18    cout << "Date and Time: " << dt.toString() << endl;
19    QDate d = dt.date();
20    cout << "Date: " << d.toString() << endl;
21    QTime t = dt.time();
22    cout << "Time: " << t.toString() << endl;
23
24    QDateTime current;
25    bintime = time((time_t *)0);
26    current.setTime_t(bintime);
27    cout << "Current date and time: "
28        << current.toString() << endl;
29    cout << "Days between: "
30        << current.daysTo(dt) << endl;
```

```
31    cout << "Seconds between: "
32           << current.secsTo(dt) << endl;
33
34    return  0;
35 }
```

이 프로그램의 출력은 다음과 같다.

```
Date and Time: Sun May 12 04:32:58 2002
Date: Sun May 12 2002
Time: 04:32:58
Current date and time: Wed Jun 28 09:51:26 2000
Days between: 683
Seconds between: 58992092
```

10～12행은 QDate와 QTime 두 객체를 사용한 QDateTime객체의 구성을 보여준다. 오직 QDate객체만 요구하는 구성자도 있는데 그것은 내부적으로 00:00:00으로 설정된 QTime객체를 구성하다.

QDate객체는 19행에서 보여준것처럼 date()호출에 의해 얻을수 있다. 새로운 QDate객체를 삽입하는데 사용하는 setDate()메쏘드도 있다. 같은 일은 time()메쏘드(21행에서 보여준것처럼)와 setTime()을 가지는 QTime객체에서도 가능하다.

시간과 날자값들을 모두 32bit체계시간값으로 설정할수 있는데 이것은 1970년 1월 1일부터의 경과초수이다. 25행에서 time()함수에 대한 체계호출은 값을 돌려주며 26행에서 QDateTime의 setTime_t()메쏘드호출은 QDateTime객체의 날자와 시간을 설정한다.

30행의 daysTo()호출은 두 날자사이의 날자수를 돌려준다. 마찬가지로 32행의 secsTo()호출은 같은 시간기간을 위한 초수를 돌려준다.

다음의 비교연산자들은 두 시간점사이의 관계를 결정한다.

```
if(datetime1 == datetime2) ...
if(datetime1 != datetime2) ...
if(datetime1 < datetime2) ...
if(datetime1 > datetime2) ...
if(datetime1 <= datetime2) ...
if(datetime1 >= datetime2) ...
```

제6절. 파일에 써넣기

다음 실례는 QFile객체를 사용하여 새로운 파일을 만들고 거기에 2행의 본문을 써넣는다.

```
1 /* writefile.cpp */
2 #include <qfile.h>
3
4 int main(int argc,char **argv)
5 {
6     char line1[] = "The first line\n";
7     char line2[] = "The second line\n";
8
9     QFile qfile("rwfile.txt");
10    if(qfile.open(IO_WriteOnly)) {
11        for(int i=0; i<strlen(line1); i++)
12            qfile.putch(line1[i]);
13        for(int i=0; i<strlen(line2); i++)
14            qfile.putch(line2[i]);
15        qfile.close();
16    }
17
18    return 0;
19 }
```

9행의 구성자는 디스크구동기의 참고를 만들지 않고 QFile객체를 만든다. 이 점에서 파일은 존재할수도 존재하지 않을수도 있다.

> 알아두기: QFile클라스는 QIODevice클라스를 계승하므로 입력과 출력에 사용될수 있다. 대체로 QDataStream이나 QTextStream객체를 사용하는것은 QFile객체를 직접 사용하는것보다 간단한다.

10행의 open()메쏘드는 파일을 써넣기전용으로 열고 성공이면 TRUE를 돌려주고 실패이면 FALSE를 돌려준다. 파일을 써넣기전용으로 여는 경우 파일이 존재하지 않으면 새로 만들고 길이를 0으로 설정한다. 파일열기방식은 표 16-1에 보여준다. 이 기발들은 OR연산자에 의해 결합할수 있다. 실례로 완충없이 써넣기 위하여 파일을 IO_Raw | WriteOnly로서 열수 있다.

표 16-1. 파일을 여는데 사용할수 있는 방식들

방식	설명
IO_Raw	파일이 완충없이 열린다. 기정은 완충기를 사용하는것이다.
IO_ReadOnly	파일이 읽기용으로 열린다. 파일이 존재하지 않으면 창조된다.

IO_WriteOnly	파일이 써넣기전용으로 열린다. 파일은 0길이로 잘리거나 또는 새로운 0길이파일이 만들어진다.
IO_ReadWrite	파일이 읽기 및 써넣기용으로 열린다. 현존파일은 잘리우지 않는다. 존재하지 않는 파일이 만들어진다.
IO_Append	파일은 써넣기용으로 열리며 다음 써넣기위치는 파일끝이다. 현존파일은 잘리지 않는다. 존재하지 않는 파일이 만들어진다.
IO_Truncate	파일은 0길이로 잘리운다.
IO_Translate	파일들은 DOS형식으로 량방향으로 변환된다. 행바꾸기문자쓰기는 복귀와 행바꾸기문자를 모두 파일에 써넣어는다. 복귀와 행바꾸기문자는 하나의 행바꾸기문자로 변환된다.

모든 쓰기는 한번에 1byte씩 진행된다. 12행과 14행의 putch()메쏘드는 파일에 한번에 1개 문자를 쓴다. 15행의 close()메쏘드는 써넣기가 수행될 때 호출되여야 하며 이것은 완충기의 자료를 디스크에 내보낸다.

제7절. 파일의 읽기

다음 실례는 QFile객체의 메쏘드들을 사용하여 ASCII파일로부터 본문행들을 읽어들인다.

```
1 /* readfile.cpp */
2 #include <qfile.h>
3 #include <iostream.h>
4
5 int main(int argc,char **argv)
6 {
7       char line[80];
8
9       QFile qfile("rwfile.txt");
10      if(qfile.open(IO_ReadOnly)) {
11              while(!qfile.atEnd()) {
12                      if(qfile.readLine(line,strlen(line)) > 0)
13                              cout << line;
14              }
15              qfile.close();
16      }
17
18      return 0;
19 }
```

파일은 10행에서 읽기전용으로 열린다. 11행의 메소드 atEnd()는 파일끝에 이를 때까지 FALSE를 계속 돌려준다. 12행의 readLine()호출은 파일로부터 한행의 본문을 읽어들여 line문자배렬에 보관한다. readLine()메소드는 실제로 읽어들인 문자수를 돌려준다. 읽어들이는 최대문자수는 둘째 인수 즉 line배렬의 용량에 의해 결정된다. 오유가 있으면(또는 파일끝에 이르면) -1이 돌아온다.

다른 메소드들을 사용하여 QFile객체를 통하여 직접 파일로부터 자료를 읽을수 있다. 문자들은 getch()를 호출하여 한번에 하나씩 읽을수 있다. 이 함수는 int값으로서 ASCII문자를 돌려주거나 오유가 있으면(또는 파일끝에 이르면) -1을 돌려준다. 다음 메소드는 본문이 아닌 자료블로크를 읽어들이는데 사용된다.

```
int count;
char block[1024];
count = qfile.readBlock(block,1024);
```

이 메소드는 파일로부터 직접 블로크에 1024byte까지 전송한다. 돌림값은 읽어들인 실제바이트수이며 오유조건이 있으면 -1을 돌려준다.

제8절. 파일에 대한 본문출력

파일에 대한 본문출력은 QTextStream객체에 출력QFile객체를 포함하여 간단히 실현한다. 다음 실례는 파일에 세행의 본문을 써넣는다.

```
1 /* streamtextout.cpp */
2 #include <qfile.h>
3 #include <qtextstream.h>
4
5 int main(int argc,char **argv)
6 {
7       char line1[] = "The first line";
8       char line2[] = "The second line";
9
10      QFile qfile("rwfile.txt");
11      if(qfile.open(IO_WriteOnly)) {
12              QTextStream stream(&qfile);
13              stream << line1 << endl;
14              stream << line2 << " and a bit more" << endl;
15              stream << "Multiplying " << 34 << " by " << 86
16                      << " gives " << 34*86 << endl;
17              qfile.close();
18      }
```

```
19
20     return 0;
21 }
```

10행과 11행에서 QFile객체가 만들어지고 출력파일이 열린다. 12행에서 QFile객체를 포함하여 QTextStream객체를 만든다. 파일이 출력용으로 열리기때문에 QTextStream클라스의 출력메쏘드들과 연산자들은 자료를 실제로 써넣는데 사용될수 있다.

이 실례에서 연산자는 흐름에 본문을 출력한다. 15행과 16행의 출력지령들은 옹근수값들을 문자렬로 변환한 다음 파일에 써넣는다. 결과는 다음과 같다.

```
The first line
The second line and a bit more
Multiplying 34 by 86 gives 2924
```

제9절. 파일로부터 본문읽기

파일로부터 한 행의 본문을 읽으려면 readLine()를 사용한다. 그러나 ASCII자료를 읽어들여 문자로부터 수값형식으로 변환하려고 하고 파일형을 알고있다면 QTextStream을 사용할수 있다. 다음은 파일에서 형식화된 본문의 실례이다.

```
orange 255 127 80 72.81 J
```

다음 실례는 본문행을 읽어들이고 행의 매 단어를 적당한 자료형으로 변환한 다음 입력행의 형식을 다시 만들어서 자료를 현시한다.

```
1 /* streamtextin.cpp */
2 #include <qfile.h>
3 #include <qtextstream.h>
4 #include <iostream.h>
5
6 int main(int argc,char **argv)
7 {
8     QString name;
9     int r;
10    int g;
11    int b;
12    double percent;
13    char code;
14
15    QFile qfile("strext.txt");
16    if(qfile.open(IO_ReadOnly)) {
17          QTextStream stream(&qfile);
```

```
18            stream >> name;
19            stream >> r >> g >> b;
20            stream >> percent;
21            stream >> code;
22            cout << name << " "
23                  << r << " " << g << " " << b << " "
24                  << percent << " " << code << endl;
25            qfile.close();
26      }
27
28      return 0;
29 }
```

파일은 15행과 16행에서 QFile객체로서 읽기전용으로 열린다. QTextStream객체는 17행에서 QFile의 용기로서 만들어진다. 18행의 >>연산자는 공백이 나타날 때까지 모든 문자들을 읽어들이고 name이라는 QString객체에 그것들을 보관한다. 19행에서 하나의 명령문에 의해 3개의 int값을 읽어들인다. 이 값들은 3개의 명령문에 의하여 읽어들일수 있는데 간단히 >>연산자들을 리용하여 1개 명령문으로 읽어들일수 있다. 20행과 21행에서 double과 char값들을 읽어들인 다음 22～24행에서 원래의 입력행이 표준출력에 다시 표시된다.

요 약

이 장에서 설명한 클라스들은 모두 매우 쓸모있다. 어떠한 크기의 프로그람이라도 이 장에서 설명한 거의 모든것을 요구할것이다. 이 장에서는 다음과 같은것을 설명하였다.

- QString클라스들은 매우 유연하고 다른 클라스에 의해 많이 사용되며 Qt와 KDE쏘프트웨어개발에서 가장 기초적인 클라스들중의 하나이다.
- 모든 클라스들은 QObject를 계승하므로 모든 객체에서 사용할수 있는 내부시계가 있다. 그것은 기동하고 정지할수 있고 지정된 간격으로 실행되도록 설정할수 있다. 그것은 사건을 발생하고 끝날 때 프로그람에 통지한다.
- 클라스 QDate, QTime 그리고 QDateTime는 달력과 시간계산에 사용할수 있는 많은 메쏘드들을 제공한다.
- QFile객체는 파일에 대하여 량방향으로 임의의 종류의 자료를 읽고쓰는데 사용될수 있다.
- QTextStream객체는 파일에 대하여 량방향으로 형식화된 자료를 읽고 쓰는데 사용될수 있다.

다음 장은 유니코드와 QChar클라스에 대하여 설명한다. QChar클라스는 이 장에서 설명한 QString클라스와 같이 기초적인 클라스이다. Qt와 KDE의 문자렬들과 문자렬조작기능들이 유니코드에 기초하고있으므로 응용프로그람에서 복합변환은 매우 단순한 과정이다.

제17장. 국제화와 환경구성

학습내용

- 원천본문을 번역용으로 만들기
- 번역된 문자렬본문의 조작
- 한개 응용프로그람을 위한 다국어번역파일의 만들기
- 유니코드와 그 조작방법을 리해하기
- 응용프로그람을 사용자정의하기 위한 환경설정값들을 정의하는 방법

동의어 i18n은 i-eighteen-letters-n의 략어로서 국제화(internationalization)라는 단어를 표시한다. KDE응용프로그람의 국제화판을 아주 간단히 작성하게 하는 편의프로그람들이 KDE개발체계에 구축되였다. 프로그람을 작성할 때 몇가지 기본규칙을 따르면 응용프로그람을 위한 번역표준비는 간단한 과정으로 된다. 일단 표를 만들면 프로그람은 실행을 시작할 때 자체로 번역한다. 또한 어떤 번역을 임의의 다른 번역으로 쉽게 넘길뿐아니라 앞으로 쏘프트웨어에 대한 변경을 동반하는 번역문의 갱신을 편리하게 해주는 봉사프로그람들도 있다.

또한 이 장에서는 매개 사용자가 응용프로그람을 사용자정의할수 있게 하는 환경구성파일 설정방법을 설명한다. 이 파일들을 대역적으로 관리할 때 모든 사용자는 같은 환경구성을 얻으며 국부적으로 고려할 때 매개 사용자는 개별적인 환경설정값들을 가지고 있을수 있다.

제1절. 번역가능한 응용프로그람

실행시에 다국어로 번역할수 있는 KDE응용프로그람은 아주 간단히 만들수 있다. 현시하거나 인쇄하려는 본문을 선언할 때 몇가지 규칙을 준비하고 응용프로그람에서 번역파일의 존재에 대하여 검사함으로써 응용프로그람이 자체로 임의의 언어로 번역할수 있다.

다음의 실례는 응용프로그람을 작성하고 번역파일들을 만드는 방법을 보여준다.

TriLang머리부파일

```
1 /* trilang.h */
2 #ifndef TRILANG_H
3 #define TRILANG_H
4
5 #include <qwidget.h>
6 #include <qlabel.h>
7
8 class TriLang: public QWidget
9 {
10    Q_OBJECT
11 public:
```

```
12    TriLang(QWidget *parent=0,const char *name=0);
13 private:
14    QLabel *label;
15    enum ButtonChoice { Rock, Paper, Scissors,
16              Clear, Exit };
17    QString emptyString;
18 private slots:
19    void slotButton(int ID);
20 };
21
22 #endif
```

이 머리부파일은 제일웃준위창문으로 표시하려는 TriLang창문부품을 정의한다. 여기에는 어떤 단추를 눌렀는가를 결정하는데 사용되는 렬거값정의도 있다.

현시하려는 모든 문자렬은 실제로 번역을 실행하는 tr()메쏘드호출내에서 선언된다. 이 실례에서 언어코드를 지령행에서 지정하면 그 이름에 의해 파일을 읽으려고 한다. 본문을 발견하면 그것을 번역한다. 본문을 발견하지 못하면 통보창문을 펼치고 프로그람은 번역없이 계속 실행된다.

TriLang

```
1 /* trilang.cpp */
2 #include <kapplication.h>
3 #include <qlayout.h>
4 #include <qhbuttongroup.h>
5 #include <qpushbutton.h>
6 #include <qmessagebox.h>
7 #include <qfileinfo.h>
8 #include "trilang.h"
9
10 int main(int argc,char **argv)
11 {
12    KApplication app(argc,argv,  "trilang");
13
14    QString lang = argv[1];
15    QString langFile = "trilang_" + lang + ".qm";
16    QFileInfo finfo(langFile);
17    if(finfo.exists()) {
```

```
18          QTranslator *qtranslator = new QTranslator(0);
19          qtranslator->load(langFile,  ".");
20          app.installTranslator(qtranslator);
21    } else {
22          QMessageBox::warning(0,  "Language File",
23                "Unable to open language file " + langFile);
24    }
25
26    TriLang *trilang = new TriLang();
27    trilang->show();
28    app.setMainWidget(trilang);
29    return  app.exec();
30 }
31
32 TriLang::TriLang(QWidget *parent,const char *name)
33    : QWidget(parent,name)
34 {
35    QPushButton *button;
36    QVBoxLayout *layout = new QVBoxLayout(this,5);
37
38    QHButtonGroup *group = new QHButtonGroup(this,  "group");
39    button = new QPushButton(tr("Rock"),group);
40    group->insert(button,Rock);
41    button = new QPushButton(tr("Paper"),group);
42    group->insert(button,Paper);
43    button = new QPushButton(tr("Scissors"),group);
44    group->insert(button,Scissors);
45    button = new QPushButton(tr("Clear"),group);
46    group->insert(button,Clear);
47    button = new QPushButton(tr("Exit"),group);
48    group->insert(button,Exit);
49    connect(group,SIGNAL(clicked(int)),
50          this,SLOT(slotButton(int)));
51    layout->addWidget(group);
52
```

```
53     emptyString = tr("-0-");
54     label = new QLabel(emptyString,this);
55     label->setAlignment(AlignVCenter | AlignHCenter);
56     layout->addWidget(label);
57
58     resize(10,10);
59     layout->activate();
60 }
61 void TriLang::slotButton(int ID)
62 {
63     switch(ID) {
64         case Rock:
65             label->setText(tr("Rock breaks scissors"));
66             break;
67         case Paper:
68             label->setText(tr("Paper covers rock"));
69             break;
70         case Scissors:
71             label->setText(tr("Scissors cut paper"));
72             break;
73         case Clear:
74             label->setText(emptyString);
75             break;
76         case Exit:
77             kapp->exit(0);
78     }
79 }
```

14～16행은 지령행에서 주어진 코드로부터 번역파일이름을 구성한다. 이것은 보통 2문자코드이다. 실례로 도이췰란드어일 때 de, 영어일 때 en을 들수 있지만 반드시 그렇게 이름을 달 필요는 없다. 그러나 많은 응용프로그람들은 다음에 제시하는 명명관례를 따른다. 즉 응용프로그람이름, 밑줄, 2문자언어코드 그리고 qm의 뒤붙이가 붙는다. 그러므로 자기의 응용프로그람이 같은 일을 할 때 보통 주의해야 한다.

17행은 파일이 존재하는가를 결정한다. 그것이 존재하면 18행에서 QTranslator객체가 만들어진다. QTranslator객체는 번역표들을 적재하고 포함할수 있다. 일단 QTranslator가 적재되면 find()메쏘드를 리용하여 알려진 단어와 구들의 모임을 번역한다. 19행의 load()메쏘드는

지령행에 의해 선택된 파일에 보관된 번역표들을 읽어들인다. 20행의 installTranslator()호출은 새로운 QTranslator를 응용프로그람의 모든 문자렬들을 번역하는데 사용하는것으로 설정한다.

번역파일이 없으면 22행의 QMessageBox가 사용자에게 통지되고 프로그람은 실행을 계속한다. 번역파일을 생략해도 프로그람을 실행할 때 오유는 없다. 그것은 정상적으로 실행되지만 어떠한 번역도 없다.

수평단추그룹은 38행의 TriLang구성자에서 생성된다. 창문부품에는 내부이름 group가 할당되고 이것은 번역되지 않는다. 내부사용을 위한 문자렬들을 번역할 필요가 없다. 번역해야할 문자렬들은 오직 현시되는 문자렬이다.

39행에서 QPushButton이 창조된다. 단추의 표식본문을 "Rock"로 정의하고 번역할수 있도록 메쏘드 tr()의 인수로 넘긴다. tr()안에서 정의되므로 이 문자렬은 번역된다. 41～47행에서 다른 단추들의 표식본문들도 모두 같은 방법으로 정의된다.

53행에서 emptyString이라는 QString객체에는 초기값이 할당된다. 또한 인용표안의 상수문자렬은 tr()에 넘기는 인수로 정의되므로 그것도 역시 번역된다.

61행에서 시작하는 slotButton()메쏘드는 본문을 여러 값들중 하나로 설정한다. 매개 경우에 상수문자렬은 tr()호출안에서 정의된다. 75행에서 setText()호출은 emptyString을 사용하지만 거기에 보관된 값은 이미 번역되였으므로 다시 번역되지 말아야 한다.

파일을 지정하지 않거나 존재하지 않는 파일을 지정함으로써 프로그람을 번역없이 실행하면 창문은 그림 17-1과 같아진다.

그림 17-1. 번역파일없이 TriLang의 실행

같은 응용프로그람이 완전히 다르게 표시되는 2개의 번역파일이 있다(다음 절들에서 자세히 설명한다). 번역파일은 현시된 모든 문자렬을 변경하지 말아야 한다.

그림 17-2는 두세개의 번역된 문자렬을 가진 같은 창문을 보여준다.

그림 17-2. 최소번역을 가지고 TriLang의 실행

현시된 모든 문자렬을 변경할수 있다. 그림 17-3은 번역파일에서 사용할수 있는 모든 문자렬을 변경한 결과를 보여준다.

그림 17-3. 완전번역을 가진 TriLang실행

제2절. 번역할수 있는 문자렬의 선언

번역파일의 적재외에 모든 응용프로그람에 대하여 번역하려는 문자렬들이 tr()메쏘드의 인수로 정의되였는가 확인하여야 한다. tr()메쏘드는 QObject를 계승하는 모든 객체에 사용할수 있다. 이것은 모든 QWidget객체들을 포함하므로 현시가능한 본문과 작업하는곳에서는 거의 어디서나 사용할수 있다.

tr()메쏘드는 QString객체를 돌려보내므로 보통 QString을 사용하는곳에서는 tr()를 사용할수 있다. 실례로 다음과 같이 QLabel을 정의한다고 하자.

```
QLabel *label = new QLabel("Select",this);
```

그것을 번역하기 위하여 tr()호출에서 인수로서 문자렬을 포함하도록 코드를 변경한다.

```
QLabel *label = new QLabel(tr("Select"),this);
```

tr()메쏘드호출이 없을수 있다. tr()에는 QObject로부터 계승하는 객체외의 문자렬정의가 있다. 그러므로 명백한 해결책은 다른 위체트의 tr()를 리용하는것이다. 실례로 함수안에서 작업하는데 후에 사용할 표식자를 만들려고 한다면 다음과 같이 QLabel의 부모창문부품을 사용한다.

```
QLabel *label = new QLabel(parent::tr("Select"),parent);
```

정적문자렬을 정의하여야 한다면 두 마크로를 사용한다. 하나는 클라스의 메쏘드안에서 사용하기 위한것이다.

```
TriLang::fstr() {
    static char *f = QT_TR_NOOP("String text");
}
```

실례로 QT_TR_NOOP마크로는 실행가능코드를 포함하지만 그것은 원천코드의 문자렬을 번역용으로 표식한다. 이 문자렬은 그것이 TriLang클라스의 성원인 메쏘드안에서 정의되므로 그 클라스의 성원으로 번역된다. 클라스의 밖에서 정적선언을 할 필요가 있다면 약간 다른 마크로가 요구된다.

```
static char *f =  QT_TRANSLATE_NOOP("TriLang","String text");
```

정의가 클라스밖에 있으므로 클라스이름도 요구된다. 이것은 이 장의 나중에 설명하는것처럼 클라스이름이 번역과정에 대한 건으로서 내적으로 사용되기때문이다. 어떠한 클라스의 이름이든 사용할수 있다. 또한 이 마크로는 본문을 번역용이라고 표식하는것외에 다른

일은 하지 않는다.

제3절. 번역된 문자렬에 대한 조작

어떠한 방법으로 문자렬들을 형식화하려고 한다면 항상 QString을 사용해야 한다. 현시가능한 문자렬들을 조작하는데 절대로 char배렬이나 QCString클라스를 사용하지 말아야 한다. 개별적인 문자들을 리용하려 한다면 유니코드문자를 리용하는데 필요한 모든 기능을 제공하는 QChar를 사용하여야 한다.

자료를 문자렬로 형식화하려면 QString의 arg()성원함수를 사용하여야 한다. 실례로 int와 double값을 리용하여 다음의 행을 현시하는 경우를 고찰하자.

```
Step 3 is 34.6 percent complete.
```

다음은 이 문자렬을 형식화하는 방법이다.

```
QString report;
report = tr("Step %1 is %2 percent complete.  ").arg(step).arg(percent);
```

이렇게 번역을 하면 제시된 문자렬은 다음과 같다.

```
"Step %1 is %2 percent complete. "
```

번역은 값들을 다른 순서로 배치되도록 요구할수 있다. 즉 필요하다면 값들을 다시 배렬할 권한을 번역기에 줄수 있다. 실례로 명령문은 다음과 같이 간단히 쓸수 있다.

```
"Completion now at %2 percent of step %1"
```

제4절. 번역파일의 구축

일단 응용프로그람을 작성하였고 모든 문자렬들이 번역용으로 적당히 정의되였다면 번역파일들을 구축할수 있다.

findtr편의프로그람은 번역하려는 모든 문자렬을 포함하는 파일을 만드는데 사용된다. 이 장에서 사용하는 실례에 영어로 번역되지 않은 판이 있고 2개의 다른 판들은 이 실례를 위해 제안되였다. 다음 지령을 입력하여 초기파일들을 만든다.

```
findtr *.cpp *.h >trilang.po
cp trilang.po trilang_en.po
cp trilang.po trilang_sh.po
cp trilang.po trilang_sl.po
```

3개의 모든 번역파일들은 꼭같은 내용으로 시작된다. findtr편의프로그람은 선두에 머리부정보를 가지는 파일을 만들며 파일의 동작부는 다음과 같다.

```
#: trilang.cpp:53
msgid "TriLang::-0-"
msgstr ""
```

```
#: trilang.cpp:45
msgid "TriLang::Clear"
msgstr ""
#: trilang.cpp:47
msgid "TriLang::Exit"
msgstr ""
#: trilang.cpp:68
msgid "TriLang::Paper covers rock"
msgstr ""
#: trilang.cpp:41
msgid "TriLang::Paper"
msgstr ""
#: trilang.cpp:65
msgid "TriLang::Rock breaks scissors"
msgstr ""
#: trilang.cpp:39
msgid "TriLang::Rock"
msgstr ""
#: trilang.cpp:71
msgid "TriLang::Scissors cut paper"
msgstr ""
#: trilang.cpp:43
msgid "TriLang::Scissors"
msgstr ""
```

msgid문자렬은 번역을 찾는데 쓰이는 문자렬이다. 매개 msgid항목은 원시프로그람에 나타나는 문자렬이다. 매개 문자렬뒤에는 그것을 포함하는 클라스의 이름과 2개의 두점이 놓인다. msgstr는 번역문자렬을 포함한다. 이 실례에서처럼 번역하려는 문자렬이 없으면 번역은 진행되지 않는다. 모든 파일들은 같은 내용으로 시작되며 오직 msgstr항목들만 변경되여야 한다. 실례로 파일 trilang_sh.po는 다음과 같은것을 포함한다.

```
#: trilang.cpp:53
msgid "TriLang::-0-"
msgstr "***”
#: trilang.cpp:45
msgid "TriLang::Clear"
msgstr "Erase”
```

```
#: trilang.cpp:47
msgid "TriLang::Exit"
msgstr ""
#: trilang.cpp:68
msgid "TriLang::Paper covers rock"
msgstr "Sheet covers rock"
#: trilang.cpp:41
msgid "TriLang::Paper"
msgstr "Sheet"
#: trilang.cpp:65
msgid "TriLang::Rock breaks scissors"
msgstr ""
#: trilang.cpp:39
msgid "TriLang::Rock"
msgstr ""
#: trilang.cpp:71
msgid "TriLang::Scissors cut paper"
msgstr "Scissors cut sheet"
#: trilang.cpp:43
msgid "TriLang::Scissors"
msgstr ""
```

클라스이름은 msgstr에서 번역된 문자렬의 부분으로 포함되지 않는다. 그것은 오직 번역과 문자렬을 대조하기 위한 문자렬로서만 msgid의 부분으로 포함된다. 또한 번역하지 말아야 할 항목은 포함할 필요가 없다.

일단 .po파일들을 만들고 편집하였으면 그것들을 응용프로그람이 사용하는 .qm파일들로 바꾸어야 한다. 편의프로그람 msg2qm은 실례의 3개 파일을 다음과 같이 변환한다.

```
msg2qm trilang_en.po trilang_en.qm
msg2qm trilang_sh.po trilang_sh.qm
msg2qm trilang_sl.po trilang_sl.qm
```

프로그람을 실행할 때 적당한 .qm파일을 읽어들이고 리용한다. 어떠한 방법으로 번역을 변경하려면 간단히 .po파일을 편집하고 그것으로 새로운 .qm파일을 만든다.

자기 프로그람을 변경하여 거기에 포함된 문자렬들을 변경하면 msg2qm을 기동하지 말아야 한다. mergetr편의프로그람은 이미 수행한 작업을 보관하고 거기에 정보를 결합할수 있게 하므로 새로운 .po파일을 생성하고 다음과 같이 변경을 결합해야 한다.

```
findtr *.cpp *.h >trilang.po
```

mergetr trilang_en.po trilang.po

mergetr trilang_sh.po trilang.po

mergetr trilang_sl.po trilang.po

이것은 이전의 모든 정보와 결합된 변경이 포함되여있는 3개의 .po파일을 제공한다. 항목들중 하나가 원래의 코드에서 변경되여 이전에 번역한 문자렬에 대한 변경을 요구한다면 이전의 번역은 주해로서 포함된다.

제5절. 유니코드와 QChar

KDE와 Qt의 표준문자모임은 유니코드이다. 매개 문자는 16bit로 표시된다. 이것은 문자가 65,536개 있을수 있다는것을 의미한다. 이것은 세계의 모든 언어의 문자들을 다 포함할수 있는 충분한 공간이다.

> 알아두기: 유니코드는 거기에 포함된 모든 문자들은 앞으로도 당분간 그대로 리용할것이다. 심지어 유니코드에는 완전한 Klingon자모들도 포함한다. 그러나 그리 멀지 않은 미래에 새로운 문자들을 포함시키기 위하여 매개 유니코드문자를 32bit로 확장할 필요가 있다.

0～127의 유니코드값들은 ASCII와 같은 문자를 표시한다(거기서 매개 문자는 0～127범위의 값이다). 이것은 ASCII에서 유니코드에로의 문자변환을 하지만 유니코드로부터 ASCII에로의 변환은 약간의 변경을 요구한다. 이것은 ASCII로 프로그람을 작성하고 필요할 때 유니코드로 자동변환하게 한다.

본문을 리용할 때 흔히 공백, 점, 대소문자 그리고 다른 문자속성들을 인식할 필요가 있다. ASCII에는 문자가 적으므로 간단하다. 유니코드는 새로운 문자를 언제나 추가할수 있다. 이것을 취급하기 위하여 매개 유니코드문자에는 속성들의 모임이 할당된다. 프로그람은 문자의 몇가지 구체적인 속성을 검사할수 있고 말하자면 그것이 대문자인가, 수자인가를 결정할수 있다. 매개 범주는 2문자코드로 표시된다. 표 17-1에서는 표준범주를 보여준다. 표준범주는 언어학에 기원을 둔 문자의 기본본성에 대한 서술이다. 표 17-2에서는 정보범주를 보여준다. 이것은 기본적인 언어학적기원외에 문자에 대한 설명이다.

표 17-1. 유니코드표준범주

이름	코드	이름	코드
Mark_Enclosing	M	Other_	C
Mark_NonSpacing	e	NotAs	n
Mark_SpacingCombining	M	signed	C
Number_Letter	n	Other_	o
Number_Other	M	Private	C
Other_Control	c	Use	s

이름	코드	이름	코드
Other_Format	Nl No Cc Cf	Other_Surrogate Separator_Line Separator_Paragraph Separator_Space	Zl Zp Zs

표 17-2. 유니코드정보범주

이름	코드	이름	코드
Letter_Uppercase Letter_Lowercase Letter_Titlecase Letter_Modifier Letter_Other Punctuation_Connector Punctuation_Dash Punctuation_Open	Lu Ll Lt Lm Lo Pc Pd Ps	Punctuation_Close Punctuation_InitialQuote Punctuation_FinalQuote Punctuati	Pe Pi Pf Po Sm Sc Sk So

이름	코드	이름	코드
		on_Other Symbol_Math Symbol_Currency Symbol_Modifier Symbol_Other	

유니코드로 작업하면 문자가 영문자, 수자, 점, 또는 어떤 종류의 공백인가를 결정하는것이 C에서 ASCII용으로 정의되였던 전통적인 isspace()와 isupper()를 사용하는것보다 좀 더 힘들다는것을 의미한다. 그것을 간단화하기 위하여 클라스는 단일한 유니코드문자를 포함하고 그것을 조작하는 많은 메쏘드들을 제공하며 그 특성에 대한 정보를 제공한다.

다음 구성자들은 QChar객체를 만든다.

QChar()

QChar(char c)

QChar(uchar c)

QChar(uchar cell, uchar row)

QChar(const QChar &c)

QChar(ushort rc)

QChar(short rc)

QChar(uint rc)

QChar(int rc)

이것들은 모두 16bit유니코드문자로서 넘긴 수값을 리용한다. 어떤 인수도 제공되지 않으면 유니코드 null문자 0x0000이 사용된다. 8bit값을 받아들이는 2개 구성자는 그 값이 ASCII문자라고 가정하고 맨 웃자리바이트를 0으로 설정한다. 2개의 8bit값을 요구하는 구성자는 제일아래자리바이트의 cell값을 사용하며 제일웃자리바이트로서 row를 리용한다.

다음 Boolean메쏘드들은 문자의 기본적인 특성을 결정하는데 쓰인다.

```
bool isDigit() const
bool isLetter() const
bool isLetterOrNumber() const
bool isMark() const
bool isNull() const
bool isNumber() const
bool isPrint() const
bool isPunct() const
bool isSpace() const
```

매개 메쏘드는 문자의 특성을 보고 돌림값을 결정한다. 실례로 문자가 어떠한 언어의 수자이면 즉 표 17-1로부터 Number_Letter 또는 Number_Other이면 isNumber()는 TRUE를 돌려준다.

자세한 정보를 요구한다면 다음 메쏘드는 표 17-1과 17-2에서 이름지어진 매개 형들의 항목을 포함하는 렬거형을 돌려준다.

```
Category category() const
```

가장 일반적인 문자변환은 대소문자의 변환으로서 다음의 메쏘드들로 수행할수 있다.

```
QChar lower() const
QChar upper() const
```

다음 메쏘드들은 유니코드문자의 수값을 돌려주는데 사용한다.

```
ushort unicode() const
char latin1() const
uchar &cell()
uchar cell() const
uchar &row()
uchar row() const
int digitValue() const
```

unicode()메쏘드는 문자의 16bit값을 돌려준다. latin1()메쏘드는 ASCII문자의 8bit값을 돌려주며 문자가 ASCII가 아니면 0을 돌려준다. cell값은 제일아래자리바이트이며 row값은 제일웃자리바이트이다. digitValue()메쏘드는 문자를 돌려주지 않으며 문자가 수자이면 수값을 int로서 돌려준다.

제6절. 환경구성

KConfig클라스는 프로그람의 한 기동으로부터 다음번 기동할 때까지 환경설정을 보관하는데 사용한다. 환경설정은 특별한 형식으로 본문파일에 보관된다. 매개 설정은 마당단어와 값의 쌍으로 보관되고 이 쌍들은 그룹들로 나누일수 있다. 다음 실례는 이동하고 크기변경할수 있는 창문을 현시함으로써 이것이 어떻게 작업하는가 보여주며 지어 프로쎄스를 멈추고 다시 기동할 때 그 크기와 위치를 얻는 방법을 보여준다.

Remember머리부파일

```
1 /* remember.h */
2 #ifndef REMEMBER_H
3 #define REMEMBER_H
4
5 #include <qwidget.h>
6
7 class Remember: public QWidget
8 {
9 public:
10     void configure();
11 private:
12     QSize windowSize;
13     QPoint windowPosition;
14 protected:
15     virtual void paintEvent(QPaintEvent *);
16     virtual void resizeEvent(QResizeEvent *);
17     virtual void moveEvent(QMoveEvent *);
18     virtual void closeEvent(QCloseEvent *);
19 };
20
21 #endif
```

Remember창문부품은 응용프로그람의 제일웃준위창문이다. 12행과 13행에서 windowSize와 windowPosition값들은 창문에 대하여 계속 갱신되는 위치정보를 보관한다.

Remember

```
1 /* remember.cpp */
2 #include <kapplication.h>
3 #include <kconfig.h>
```

```
4 #include <qpainter.h>
5 #include <qdir.h>
6 #include "remember.h"
7
8 int main(int argc,char **argv)
9 {
10     KApplication app(argc,argv,"remember");
11     Remember remember;
12     remember.configure();
13     remember.show();
14     app.setMainWidget(&remember);
15     return  app.exec();
16 }
17 void Remember::paintEvent(QPaintEvent *)
18 {
19     QPainter p(this);
20
21     p.setWindow(0,0,300,300);
22     p.drawRoundRect(50,50,200,200,30,30);
23     p.setBrush(QColor("white"));
24     p.drawEllipse(100,100,100,100);
25 }
26 void Remember::resizeEvent(QResizeEvent *event)
27 {
28     windowSize = event->size();
29 }
30 void Remember::moveEvent(QMoveEvent *event)
31 {
32     windowPosition = event->pos();
33 }
34 void Remember::closeEvent(QCloseEvent *event)
35 {
36     QString str;
37
38     KConfig *config =
```

```
39              new KConfig(QDir::homeDirPath() + "/.remember");
40      config->setGroup("Geometry");
41      config->writeEntry("width",
42      str.setNum(windowSize.width()));
43      config->writeEntry("height",
44      str.setNum(windowSize.height()));
45      config->writeEntry("xPosition",
46      str.setNum(windowPosition.x()));
47      config->writeEntry("yPosition",
48      str.setNum(windowPosition.y()));
49      config->sync();
50      delete config;
51
52      event->accept();
53 }
54 void Remember::configure()
55 {
56      int width;
57      int height;
58      int x;
59      int y;
60      QString str;
61
62      KConfig *config =
63              new KConfig(QDir::homeDirPath() + "/.remember");
64      config->setGroup("Geometry");
65      width = config->readNumEntry("width",100);
66      height = config->readNumEntry("height",100);
67      x = config->readNumEntry("xPosition",10);
68      y = config->readNumEntry("yPosition",10);
69
70      windowSize = QSize(width,height);
71      windowPosition = QPoint(x,y);
72
73      resize(windowSize);
```

```
74    move(windowPosition);
75 }
```

프로그람의 main()함수는 8행에서 시작한다. 그것은 Remember창문부품을 창조하여 제일웃준위창문으로서 사용한다. 12행에 창문이 현시되기전에 환경구성자료를 읽어들여 창문크기와 위치를 설정하기 위한 configure()메쏘드호출이 있다.

17행의 paintEvent()메쏘드는 QPainter객체를 사용하여 그림의 크기를 실제창문크기에 맞게 자동적으로 조절한다. 그러기 위하여 setWindow()를 호출하여 너비와 높이의 최대값들을 설정하고 그 치수를 사용하여 도형으로 직4각형을 채운다. 그림 17-4에서는 각이한 크기의 Remember창문을 보여준다.
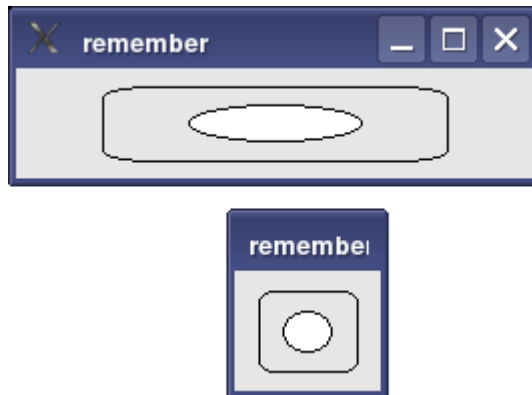
그림 17-4. 여러가지 창문의 크기와 위치기억

Remember창문부품은 현재크기와 장소의 리력을 가지고있다. 26행의 resizeEvent()메쏘드는 창문크기가 변화될 때마다 호출되고 30행의 moveEvent()는 창문이 이동할 때마다 호출된다. 두 메쏘드는 항상 windowSize와 windowPosition에 보관된 값을 갱신한다.

34행의 closeEvent()메쏘드는 응용프로그람이 닫길 때마다 호출된다. 현재 환경구성설정을 보관하기 위하여 사용자의 홈등록부에서 .remember라는 파일을 열어 KConfig객체를 만든다. 정적메쏘드 QDir::homeDirPath()는 현재사용자의 홈등록부의 완전경로를 돌려주는데 이것은 사용자마다 이 프로그람용으로 개별적인 환경구성파일을 가지고있다는것을 의미한다.

40행의 setGroup()호출은 환경구성정보를 포함하는 파일안의 그룹이름을 지정한다. 파일에 그룹이 여러개 있을수 있고 같은 마당값은 여러개의 그룹에 나타날수 있다. 임의의 시각에 KConfig객체는 오직 1개의 그룹만 호출할수 있으나 setGroup을 호출하여 한 그룹에서 다른 그룹으로 절환할수 있다. 41～48행에서 매개 writeEntry()호출은 환경구성에 단일항목을 써넣는다. 매개 항목은 값문자렬과 그것을 발견하는데 사용되는 마당으로 이루어진다. 이 실례에 의하여 씌여진 본문은 다음과 같다.

```
[Geometry]
height=170
width=406
```

xPosition=346

yPosition=130

파일에서 그룹이름들은 중괄호에 포함되고 그뒤에 오는 모든 마당-값의 쌍들은 그룹안에 있다.

writeEntry()메쏘드는 파일에 직접 써넣지 않는다. KConfig객체가 만들어지면 파일(있다면)을 읽고 그로부터 모든 정의를 적재한다. 41～48행에서 writeEntry()호출은 간단히 마당-값들의 RAM상주목록을 갱신한다. 그것은 49행에서 sync()메쏘드호출에 필요하기때문이다. sync()메쏘드는 파일에 모든것을 써넣음으로써 파일자료와 RAM자료를 동기화한다.

실제로 응용프로그람이 닫기는것을 방지하는데 closeEvent()를 사용할수 있다. 이 실례에서 52행의 accept()호출은 이 응용프로그람이 닫기요구를 받아들인다는것을 의미하므로 체계는 실행되지만 창문은 닫긴다. 이 메쏘드를 호출하지 않으면 응용프로그람은 닫기지 않고 기정 closeEvent()메쏘드가 간단히 accept()을 호출한다. 54행의 configure()호출은 이전 환경구성설정(있다면)을 읽어들이고 자료를 사용하여 현재 응용프로그람의 환경을 구성한다. KConfig객체는 62행과 63행에서 만들어지고 closeEvent()에서처럼 파일로부터 모든 환경구성정보를 읽어들인다. 64행의 setGroup()호출은 자료를 "Geometry"라는 그룹으로부터 읽어들인다. 65～68행의 readNumEntry()호출에서 마당들이 주어지고 서로 련결된 int값들을 돌려준다. 호출의 둘째 인수는 마당값이 없으면 돌려받은 기정값이다. 함수호출에서 기정값을 줌으로써 오유처리를 간단화한다. 마당이 없으면 기정값이 사용된다. 70행과 71행에서 이 값들은 응용프로그람이 닫길때 보관된다. 73행과 74행에서 resize()와 move ( )호출은 창문자체의 환경을 구성한다.

요 약

인터네트의 출현과 쏘프트웨어의 국제적인 배포와 함께 늘 쏘프트웨어를 언어에 따라 이식할수 있게 만드는것이 중요하다. KDE에 구축된 편의프로그람들은 번역할수 있는 쏘프트웨어대면부들을 만들고 유지하는 일을 대단히 쉽게 해준다. 이 장에서는 다음과 같은 이식가능항목들을 론의하였다.

• QObject의 tr()메쏘드에 대한 인수로서 상수문자렬을 포함하면 프로그람을 실행할 때 문자렬들을 번역할수 있다.

• 메쏘드밖에서 정적으로 정의된 문자렬들의 번역은 프로그람을 실행할 때 그 문자렬들을 번역용이라고 표식하는 특수마크로안에 포함할것을 요구한다.

• 간단한 편의프로그람들을 리용하여 프로그람의 원천코드를 조사하고 실제번역시에 작성자가 편집할 번역파일들을 만든다. 그때 이 파일들의 내용을 읽어들이고 실제번역메쏘드용의 대리들을 만든다.

• 환경구성자료는 환경설정을 그룹들로 구분하고 매개 그룹에 탐색가능한 마당들을 할당하는 형식으로 본문파일들에 기억될수 있다.

다음 장에서는 모든 Qt 창문부품들에 대하여 간단히 설명한다.

제18장. Qt의 창문부품

학습내용

매개 창문부품의 구성자, 필요한 머리부파일, 기초클라스들과 파생클라스들, 신호와 처리부들, 공개메쏘드들

매개 창문부품의 자그마한 실례.

창문부품은 현시가능한 창문을 포함하는 클라스이다. Qt에서 현시가능한 창문을 가진 모든 클라스는 QWidget클라스로부터 창문기능을 계승한다.

이 장에서는 매개 창문부품에 대하여 머리부파일의 이름, 모든 기초클라스들의 이름, Qt와 KDE의 모든 파생클라스들의 이름 그리고 공개메쏘드들, 처리부, 신호, 및 렬거형들을 제시한다.

매개 창문부품에 적어도 하나의 실례프로그람이 있고 매개 실례는 창문부품의 현시가능한 폼을 만든다. 일부 창문부품들은 제일웃준위창문부품으로서 지정하여 설명하며 기본창문으로 현시된다. 그러나 일부 특수창문부품들은 특정한 환경에서 포함된다.

QButton

이 창문부품은 그 모양을 바꾸고 신호를 발생함으로써 마우스사건에 응답한다.

머리부파일

```
#include <qbutton.h>
```

기초클라스

QObject QPaintDevice QWidget Qt

파생클라스

KColorButton KDialogBaseButton KDirectionButton

KDockButton_Private KIconButton KKeyButton KTabButton

KToolBarButton QCheckBox QPushButton QRadioButton

QToolButton

구성자

```
QButton(QWidget *parent = 0, const char *name = 0, WFlags f = 0);
```

메쏘드

```
int accel() const;
bool autoRepeat() const;
bool autoResize() const;
bool focusNextPrevChild(bool next);
QButtonGroup *group() const;
bool isDown() const;
```

bool isExclusiveToggle() const;

bool isOn() const;

bool isToggleButton() const;

const QPixmap *pixmap() const;

virtual void setAccel(int);

virtual void setAutoRepeat(bool);

virtual void setAutoResize(bool);

virtual void setDown(bool);

virtual void setPixmap(const QPixmap &);

virtual void setText(const QString &);

ToggleState state() const;

QString text() const;

ToggleType toggleType() const;

처리부

void animateClick();

void toggle();

신호

void clicked();

void pressed();

void released();

void stateChanged(int);

void toggled(bool);

렬거형

enum ToggleType { SingleShot, Toggle, Tristate };

enum ToggleState { Off, NoChange, On };

7장에 QButton창문부품의 실례가 있다.

QButtonGroup

이 창문부품은 수평으로 또는 수직으로 배치된 단추모임의 용기이다.

머리부파일

#include <qbuttongroup.h>

기초클라스

QFrame QGroupBox QObject QPaintDevice QWidget Qt

파생클라스

QHButtonGroup QVButtonGroup

구성자

QButtonGroup(QWidget *parent = 0, const char *name = 0);

QButtonGroup(const QString &title, QWidget *parent = 0, const char *name = 0);

QButtonGroup(int columns, Orientation o, QWidget *parent = 0, const char *name = 0);

QButtonGroup(int columns, Orientation o, const QString &title, QWidget *parent = 0,
const char *name = 0);

메쏘드

int count() const;

QButton *find(int id) const;

int id(QButton *) const;

int insert(QButton *, int id = - 1);

bool isExclusive() const;

bool isRadioButtonExclusive() const;

virtual void moveFocus(int);

void remove(QButton *);

QButton *selected();

virtual void setButton(int id);

virtual void setExclusive(bool);

virtual void setRadioButtonExclusive(bool);

신호

void clicked(int id);

void pressed(int id);

void released(int id);

이 클라스는 직접 사용되지 않는다.

7장에 QHButtonGroup와 QVButtonGroup의 실례가 있다.

QCheckBox

이 창문부품은 검사칸이다. 검사칸은 마우스에 의해 on과 off로 절환될수 있고 항상 현재상태를 현시한다.

머리부파일

#include <qcheckbox.h>

기초클라스

QButton QObject QPaintDevice QWidget Qt

구성자

QCheckBox(QWidget *parent, const char *name = 0);

QCheckBox(const QString &text, QWidget *parent, const char *name = 0)

메쏘드

bool isChecked() const;

void setChecked(bool check);

void setNoChange();

void setTristate(bool y = TRUE);

QSize sizeHint() const;

QSizePolicy sizePolicy() const;

7장에 QCheckBox창문부품을 수직 및 수평단추그룹에서 사용하는 많은 실례가 있다.

QColorDialog

이 클라스는 사용자가 색을 선택할것을 재촉하는 대화칸을 펼치는데 쓰이는 정적메쏘드들의 모임이다.

머리부파일

#include <qcolordialog.h>

기초클라스

QDialog QObject QPaintDevice QWidget Qt

메쏘드

static QRgb customColor(int);

static int customCount();

static QColor getColor(QColor, QWidget *parent = 0, const char *name = 0);

static QRgb getRgba(QRgb, bool *ok = 0, QWidget *parent = 0, const char *name = 0);

static void setCustomColor(int, QRgb);

다음 실례는 기정선택으로서 붉은색을 사용하고 사용자가 그 색을 받아들이거나 다른 색을 선택할것을 재촉한다. 대화칸은 그림 18-1에 보여준다.

```
/* showcolordialog.cpp */
#include <qapplication.h>
#include <qcolordialog.h>
int main(int argc,char **argv)
{
   QApplication app(argc,argv);
   QColor seedColor("red");
   QColor newColor;
   newColor = QColorDialog::getColor(seedColor);
   return(app.exec());
```

}
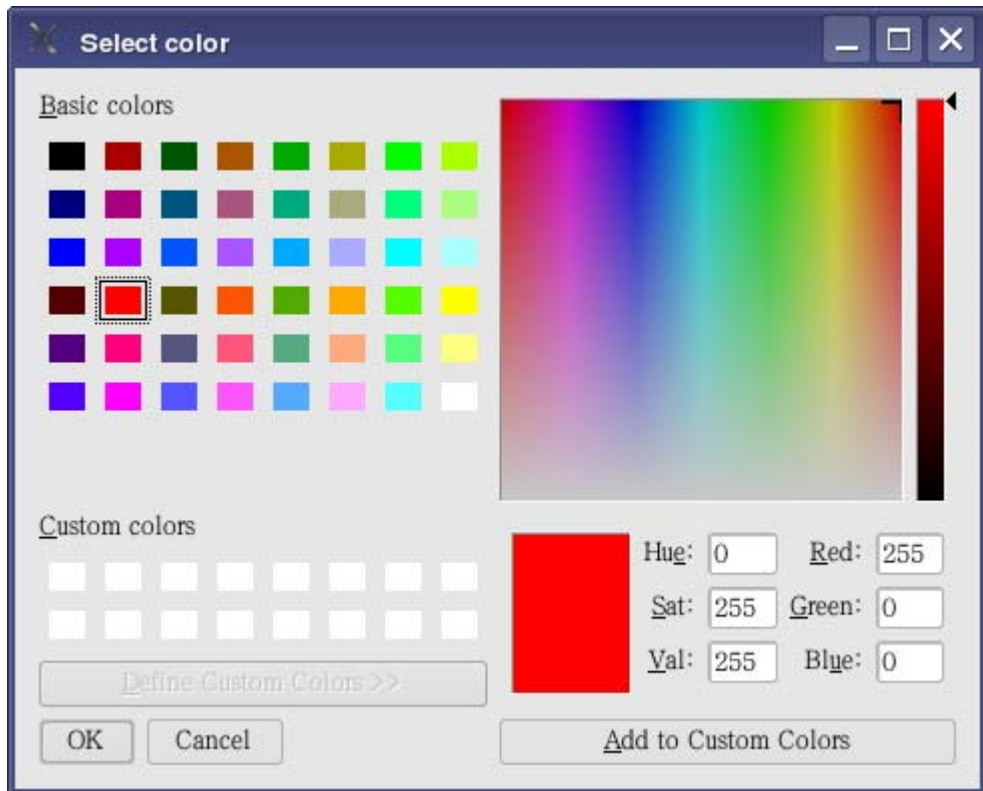
그림 18-1. QColorDialog창문부품

QComboBox

이 창문부품은 사용자가 목록으로부터 선택하는데 단추를 사용하며 늘 현재선택항목을 현시한다.

머리부파일

#include <qcombobox.h>

기초클라스

QObject QPaintDevice QWidget Qt

파생클라스

KColorCombo KComboBox KFileComboBox KFileFilter KURLComboBox

구성자

```
QComboBox(QWidget *parent = 0, const char *name = 0);
QComboBox(bool rw, QWidget *parent = 0, const char *name = 0);
```

메쏘드

```
bool autoCompletion() const;
bool autoResize() const;
```

```
void changeItem(const QString &text, int index);
void changeItem(const QPixmap &pixmap, int index);
void changeItem(const QPixmap &pixmap, const QString &text, int index);
void clear();
int count() const;
int currentItem() const;
QString currentText() const;
bool duplicatesEnabled() const;
bool eventFilter(QObject *object, QEvent *event);
void insertItem(const QString &text, int index = - 1);
void insertItem(const QPixmap &pixmap, int index = - 1);
void insertItem(const QPixmap &pixmap, const QString &text, int index = - 1);
void insertStrList(const QStrList &, int index = - 1);
void insertStrList(const QStrList *, int index = - 1);
void insertStrList(const char **, int numStrings = - 1, int index = - 1);
void insertStringList(const QStringList &, int index = - 1);
Policy insertionPolicy() const;
QLineEdit *lineEdit() const;
QListBox *listBox() const;
int maxCount() const;
const QPixmap *pixmap(int index) const;
void removeItem(int index);
virtual void setAutoCompletion(bool);
virtual void setAutoResize(bool);
virtual void setBackgroundColor(const QColor &);
virtual void setCurrentItem(int index);
void setDuplicatesEnabled(bool enable);
virtual void setEnabled(bool);
virtual void setFont(const QFont &);
virtual void setInsertionPolicy(Policy policy);
virtual void setListBox(QListBox *);
virtual void setMaxCount(int);
virtual void setPalette(const QPalette &);
virtual void setSizeLimit(int);
virtual void setValidator(const QValidator *);
QSize sizeHint() const;
```

```
int sizeLimit() const;
virtual QSizePolicy sizePolicy() const;
QString text(int index) const;
const QValidator *validator() const;
```

처리부

```
void clearEdit();
void clearValidator();
virtual void setEditText(const QString &);
```

신호

```
void activated(int index);
void activated(const QString &);
void highlighted(int index);
void highlighted(const QString &);
void textChanged(const QString &);
```

렬거형

```
enum Policy { NoInsertion, AtTop, AtCurrent, AtBottom, AfterCurrent, BeforeCurrent };
```

다음 실례는 4개의 항목을 가지는 QComboBox를 만든다. 그림 18-2는 마우스로 목록을 펼치고 둘째 항목을 선택한 후의 창문부품을 보여준다.

```
/* showcombobox.cpp */
#include <qapplication.h>
#include <qcombobox.h>
const char *list[] = { "First Selection", "Second Selection", "Third Selection", "Fourth Selection" };
int main(int argc,char **argv)
{
   QApplication app(argc,argv);
   QComboBox *combobox = new QComboBox();
   combobox->insertStrList(list,4);
   combobox->show();
   app.setMainWidget(combobox);
   return(app.exec());
}
```

그림 18-2. 4개 항목을 가지는 QComboBox창문부품

QDialog

이 창문부품은 대화칸의 기초클라스이다. 기초클라스로 사용할수 있으며 초기화할 때 창문부품들을 채워넣어 대화칸을 동적으로 구성할수 있다.

머리부파일

#include <qdialog.h>

기초클라스

QObject QPaintDevice QWidget Qt

파생클라스

KAboutDialog KAboutKDE KBugReport KColorDialog KCookieWin KDialog KDialogBase KEdFind

KEdGotoLine KEdReplace KEditToolbar KFileDialog KFileDialogConfigureDlg KFontDialog KIconDial og

KKeyDialog KLineEditDlg KOpenWithDlg KPasswordDialog KTextPrintDialog KURLRequesterDlg

KWizard KabAPI QColorDialog QfileDialog QFontDialog QInputDialog QMessageBox QPrintDialog

QTabDialog QWizard

구성자

```
QDialog(QWidget *parent = 0, const char *name = 0, bool modal = FALSE, WFlags f = 0);
```

메쏘드

```
int exec();
void hide();
void move(int x, int y);
void move(const QPoint &p);
void resize(int w, int h);
void resize(const QSize &);
int result() const;
void setGeometry(int x, int y, int w, int h);
void setGeometry(const QRect &);
void show();
```

렬거형

```
enum DialogCode { Rejected, Accepted };
```

4장에 QDialog창문부품을 사용하여 대화칸창문을 구성하는 실례들이 있다.

QFileDialog

이 대화칸은 파일의 이름과 위치를 사용자로부터 얻을 때 사용된다.

머리부파일

```
#include <qfiledialog.h>
```

기초클라스

QDialog QObject QPaintDevice QWidget Qt

구성자

```
QFileDialog(const QString &dirName,
const QString &filter = QString::null, QWidget *parent = 0, const char *name = 0,
        bool modal = FALSE);
QFileDialog(QWidget *parent = 0, const char *name = 0, bool modal = FALSE);
```

메쏘드

```
const QDir *dir() const;
QString dirPath() const;
bool eventFilter(QObject *, QEvent *);
static QString getExistingDirectory(const QString &dir = QString::null, QWidget *parent = 0,
        const char *name = 0);
static QString getExistingDirectory(const QString &dir, QWidget *parent, const char *name,
        const QString &caption);
static QString getOpenFileName(const QString &initially = QString::null,
        const QString &filter = QString::null, QWidget *parent = 0, const char *name = 0);
static QString getOpenFileName(const QString &initially, const QString &filter, QWidget *parent,
        const char *name, const QString &caption);
static QStringList getOpenFileNames(const QString &filter = QString::null,
        const QString &dir = QString::null, QWidget *parent = 0, const char *name = 0);
static QStringList getOpenFileNames(const QString &filter, const QString &dir, QWidget *parent,
        const char *name, const QString &caption);
static QString getSaveFileName(const QString &initially = QString::null,
        const QString &filter = QString::null, QWidget *parent = 0, const char *name = 0);
static QString getSaveFileName(const QString &initially, const QString &filter, QWidget *parent,
        const char *name, const QString &caption);
static QFileIconProvider *iconProvider();
bool isContentsPreviewEnabled() const;
bool isInfoPreviewEnabled() const;
Mode mode() const;
PreviewMode previewMode() const;
void rereadDir();
void resortDir();
void selectAll(bool b);
```

```
QString selectedFile() const;
QStringList selectedFiles() const;
QString selectedFilter() const;
void setContentsPreview(QWidget *w, QFilePreview *preview);
void setContentsPreviewEnabled(bool);
void setDir(const QDir &);
static void setIconProvider(QFileIconProvider *);
void setInfoPreview(QWidget *w, QFilePreview *preview);
void setInfoPreviewEnabled(bool);
void setMode(Mode);
void setPreviewMode(PreviewMode m);
void setSelection(const QString &);
void setShowHiddenFiles(bool s);
void setViewMode(ViewMode m);
bool showHiddenFiles() const;
QUrl url() const;
ViewMode viewMode() const;
```

처리부

```
void setDir(const QString &);
void setFilter(const QString &);
void setFilters(const QString &);
void setFilters(const char **);
void setFilters(const QStringList &);
void setUrl(const QUrlOperator &url);
```

신호

```
void dirEntered(const QString &);
void fileHighlighted(const QString &);
void fileSelected(const QString &);
```

렬거형

```
enum Mode { AnyFile, ExistingFile, Directory, ExistingFiles };
enum ViewMode { Detail, List };
enum PreviewMode { NoPreview, Contents, Info };
```

5장에 QFileDialog의 실례가 있다.

QFontDialog

이 클라스는 서체선택대화칸을 펼치는 정적메쏘드들로 이루어져있다.

머리부파일

#include <qfontdialog.h>

기초클라스

QDialog QObject QPaintDevice QWidget Qt

메쏘드

```
static QFont getFont(bool *ok, const QFont &def, QWidget *parent = 0, const char *name = 0);
static QFont getFont(bool *ok, QWidget *parent = 0, const char *name = 0);
```

신호

```
void fontHighlighted(const QFont &font);
void fontSelected(const QFont &font);
```

10장에 QFontDialog의 실례가 많다.

QFrame

이것은 틀로 둘러싸인 창문부품들의 기초클라스이다. 또한 빈 창문부품이므로 실례를 만들어 다른 위체트들을 채울수 있다.

머리부파일

#include <qframe.h>

기초클라스

QObject QPaintDevice QWidget Qt

파생클라스

KAboutContainer KAboutContributor KAccelMenu KAnimWidget KApplicationTree KCharSelect KCharSelectTable KColorCells KColorPatch KCombiView KDateInternalMonthPicker KDatePicker KDateTable KDockWidgetAbstractHeader KDockWidgetAbstractHeaderDrag KDockWidgetHeader KDockWidgetHeaderDrag KDockWindow KEdit KFileDetailView KFileIconView KFilePreview KFormulaToolBar KHTMLView KIconCanvas KIconView KImageTrackLabel KIntSpinBox KListBox KListView KMenuBar KPopupMenu KPopupMenu KProgress KRuler KSeparator KSplitList KStatusBarLabel KTextBrowser KToolBar KURLLabel KURLRequester QButtonGroup QCanvasView

QGrid QGroupBox QHBox QHButtonGroup QHGroupBox QIconView QLCDNumber QLabel QListBox QListView QMenuBar QMultiLineEdit QPopupFrame QPopupMenu QProgressBar QScrollView QSpinBox QSplitter QTableView QTextBrowser QTextView QVBox QVButtonGroup QVGroupBox QWellArray QWidgetStack

구성자

QFrame(QWidget *parent = 0, const char *name = 0, WFlags f = 0, bool = TRUE);

메쏘드

QRect contentsRect() const;

QRect frameRect() const;

Shadow frameShadow() const;

Shape frameShape() const;

int frameStyle() const;

int frameWidth() const;

bool lineShapesOk() const;

int lineWidth() const;

int margin() const;

int midLineWidth() const;

virtual void setFrameRect(const QRect &);

void setFrameShadow(Shadow);

void setFrameShape(Shape);

virtual void setFrameStyle(int);

virtual void setLineWidth(int);

virtual void setMargin(int);

virtual void setMidLineWidth(int);

QSize sizeHint() const;

QSizePolicy sizePolicy() const;

렬거형

enum Shape { NoFrame=0, Box=0x0001, Panel=0x0002, WinPanel=0x0003, HLine=0x0004,
    VLine=0x0005,StyledPanel=0x0006, PopupPanel=0x0007, MShape=0x000f };

enum Shadow { Plain=0x0010, Raised=0x0020, Sunken=0x0030, MShadow=0x00f0 };

7장에 QFrame창문부품을 사용하는 많은 실례가 있다.

QGrid

이 배치관리기는 내부적으로 관리되는 살창우에서의 자리표위치들에 따라 자식창문부품들의 크기와 위치를 조절한다.

머리부파일

#include <qgrid.h>

기초클라스

QFrame QObject QPaintDevice QWidget Qt

구성자

QGrid(int n, QWidget *parent = 0, const char *name = 0, WFlags f = 0);

QGrid(int n, Direction, QWidget *parent = 0, const char *name = 0, WFlags f = 0);

메쏘드

void setSpacing(int);

QSize sizeHint() const;

렬거형

enum Direction { Horizontal, Vertical };

3장에 QGrid의 실례가 많다.

QGroupBox

이 창문부품은 다른 창문부품들을 포함하고 경계선과 제목을 제공한다.

머리부파일

#include <qgroupbox.h>

기초클라스

QFrame QObject QPaintDevice QWidget Qt

파생클라스

QButtonGroup QHButtonGroup QHGroupBox QVButtonGroup QVGroupBox

구성자

QGroupBox(QWidget *parent = 0, const char *name = 0);

QGroupBox(const QString &title, QWidget *parent = 0, const char *name = 0);

QGroupBox(int columns, Orientation o, QWidget *parent = 0, const char *name = 0);

QGroupBox(int columns, Orientation o, const QString &title, QWidget *parent = 0,
 const char *name = 0);

메쏘드

void addSpace(int);

int alignment() const;

int columns() const;

Orientation orientation() const;

virtual void setAlignment(int);

virtual void setColumnLayout(int columns, Orientation o);

void setColumns(int);

void setOrientation(Orientation);

virtual void setTitle(const QString &);

QString title() const;

다음 실례는 QGroupBox창문부품을 제일웃준위창문으로 사용한다. 내용은 없지만 그림 18-3에 보여준것처럼 제목문자렬이 주어진다.

```
/* showgroupbox.cpp */
#include <qapplication.h>
#include <qgroupbox.h>
int main(int argc,char **argv)
{
    QApplication app(argc,argv);
    QGroupBox *groupbox = new QGroupBox();
    groupbox->setTitle("Group Title");
    groupbox->show();
    app.setMainWidget(groupbox);
    return(app.exec());
}
```

그림 18-3. 제일웃준위창문부품으로서 QGroupBox

QHBox

QHBox창문부품은 창문부품들을 나란히 배치하는 간단한 용기이다.

머리부파일

```
#include <qhbox.h>
```

기초클라스

QFrame QObject QPaintDevice QWidget Qt

파생클라스

KCharSelect KURLRequester QVBox

구성자

```
QHBox(QWidget *parent = 0, const char *name = 0, WFlags f = 0, bool allowLines = TRUE);
```

메쏘드

```
void setSpacing(int);
bool setStretchFactor(QWidget *, int stretch);
QSize sizeHint() const;
```

다음 실례는 제일웃준위창문부품으로서 QHBox를 사용한다. 자식창문부품으로서 4개의

QLabel창문부품을 가진다. 그림 18-4에 보여주는것처럼 매개 표식자는 5화소너비로 나란히 현시된다.

```
/* showhbox.cpp */
#include <qapplication.h>
#include <qhbox.h>
#include <qlabel.h>
int main(int argc,char **argv)
{
    QApplication app(argc,argv);
    QHBox *hbox = new QHBox();
    new QLabel("First", hbox);
    new QLabel("Second", hbox);
    new QLabel("Third", hbox);
    new QLabel("Fourth", hbox);
    hbox->setSpacing(5);
    hbox->show();
    app.setMainWidget(hbox);
    return(app.exec());
}
```

그림 18-4. QHBox에 의해 현시된 표식자들

QHButtonGroup

QHButtonGroup는 수평으로 단추들을 배치하는 용기창문부품이다.

머리부파일

```
#include <qhbuttongroup.h>
```

기초클라스

QButtonGroup QFrame QGroupBox QObject QPaintDevice QWidget Qt

구성자

```
QHButtonGroup(QWidget *parent = 0, const char *name = 0);
QHButtonGroup(const QString &title, QWidget *parent = 0,const char *name = 0);
```

7장에 QHButtonGroup를 사용한 실례프로그람이 있다.

QHeader

QHeader창문부품은 많은 렬표제의 크기와 위치를 조종하는 용기이다.

머리부파일

```
#include <qheader.h>
```

기초클라스

QObject QPaintDevice QWidget Qt

구성자

```
QHeader(QWidget *parent = 0, const char *name = 0);
QHeader(int, QWidget *parent = 0, const char *name = 0);
```

메쏘드

```
int addLabel(const QString &, int size = - 1);
int addLabel(const QIconSet &, const QString &, int size = - 1);
int cellAt(int) const;
int cellPos(int) const;
int cellSize(int) const;
int count() const;
QIconSet *iconSet(int section) const;
bool isClickEnabled(int section = - 1) const;
bool isMovingEnabled() const;
bool isResizeEnabled(int section = - 1) const;
QString label(int section) const;
int mapToActual(int) const;
int mapToIndex(int section) const;
int mapToLogical(int) const;
int mapToSection(int index) const;
virtual void moveCell(int, int);
void moveSection(int section, int toIndex);
int offset() const;
Orientation orientation() const;
void removeLabel(int section);
void resizeSection(int section, int s);
int sectionAt(int pos) const;
int sectionPos(int section) const;
int sectionSize(int section) const;
virtual void setCellSize(int, int);
```

```
virtual void setClickEnabled(bool, int section = - 1);
virtual void setLabel(int, const QString &, int size = - 1);
virtual void setLabel(int, const QIconSet &, const QString &, int size = - 1);
virtual void setMovingEnabled(bool);
virtual void setOrientation(Orientation);
virtual void setResizeEnabled(bool, int section = - 1);
void setSortIndicator(int section, bool increasing = TRUE);
virtual void setTracking(bool enable);
QSize sizeHint() const;
QSizePolicy sizePolicy() const;
bool tracking() const;
```

처리부

```
virtual void setOffset(int pos);
```

신호

```
void clicked(int section);
void indexChange(int section, int fromIndex, int toIndex);
void moved(int, int);
void pressed(int section);
void released(int section);
void sectionClicked(int);
void sizeChange(int section, int oldSize, int newSize);
```

다음 실례는 본문길이가 서로 다른 4개 렬의 QHeader를 만든다. 매개 렬표제의 크기는 마우스로 조절할수 있다. 표제는 본문이 명백하지 않아도 거기에 할당된 너비를 유지한다. 그림 18-5에 보여주는것처럼 표제부는 창문의 오른쪽끝을 넘을수도 있고 하나의 렬표제가 다른것과 겹치도록 크기를 변경할수 있다. 또한 렬표제는 본문을 현시하는데 필요한 크기이상으로 확장될수 있다. QHeader창문부품에 의해 창조된 그룹은 표제의 아래에 있는 렬들의 크기와 상태를 유지하는데 쓰일수 있다.

```
/* showheader.cpp */
#include <qapplication.h>
#include <qheader.h>
int main(int argc,char **argv)
{
    QApplication app(argc,argv);
    QHeader *header = new QHeader();
    header->addLabel("Column One");
    header->addLabel("Two");
```

```
    header->addLabel("Three");
    header->addLabel("Fourth Column");
    header->show();
    app.setMainWidget(header);
    return(app.exec());
}
```
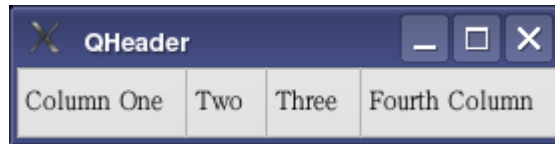
그림 18-5. 4렬의 표제를 포함하는 QHeader창문부품

QHGroupBox

QHGroupBox는 수평행안에 창문부품모임을 배치하는 용기창문부품이다.

머리부파일

```
#include <qhgroupbox.h>
```

기초클라스

QFrame QGroupBox QObject QPaintDevice QWidget Qt

구성자

```
QHGroupBox(QWidget *parent = 0, const char *name = 0);
QHGroupBox(const QString &title, QWidget *parent = 0, const char *name = 0);
```

다음 실례는 QHGroupBox창문부품안에 4개 표식자를 포함한다. 그림 18-6에 보여주는것처럼 QHGroupBox창문부품은 QFrame로부터 계승되므로 포함된 창문부품들의 둘레에 경계선을 현시하고 마음대로 제목을 현시할수 있다.

```
/* showhgroupbox.cpp */
#include <qapplication.h>
#include <qhgroupbox.h>
#include <qlabel.h>
int main(int argc,char **argv)
{
    QApplication app(argc,argv);
    QHGroupBox *hgroupbox = new QHGroupBox();
    new QLabel("First", hgroupbox);
    new QLabel("Second", hgroupbox);
    new QLabel("Third", hgroupbox);
    new QLabel("Fourth", hgroupbox);
```

```
    hgroupbox->setTitle("Group Box Title");
    hgroupbox->show();
    app.setMainWidget(hgroupbox);
    return(app.exec());
}
```
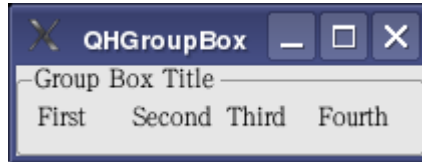
그림 18-6. QHGroupBox에 의해 포함된 4개 단추

QIconView

QIconView창문부품은 그림기호모임을 현시하며 사용자가 선택할수 있게 한다.

머리부파일

#include <qiconview.h>

기초클라스

QFrame QObject QPaintDevice QScrollView QWidget Qt

파생클라스

KFileIconView KIconCanvas KIconView

구성자

QIconView(QWidget *parent = 0, const char *name = 0, WFlags f = 0);

메쏘드

```
Arrangement arrangement() const;
bool autoArrange() const;
virtual void clear();
virtual void clearSelection();
uint count() const;
QIconViewItem *currentItem() const;
void ensureItemVisible(QIconViewItem *item);
bool eventFilter(QObject *o, QEvent *);
QIconViewItem *findFirstVisibleItem(const QRect &r) const;
QIconViewItem *findItem(const QPoint &pos) const;
QIconViewItem *findItem(const QString &text) const;
QIconViewItem *findLastVisibleItem(const QRect &r) const;
QIconViewItem *firstItem() const;
```

```
int gridX() const;
int gridY() const;
int index(const QIconViewItem *item) const;
virtual void insertItem(QIconViewItem *item, QIconViewItem *after = 0L);
virtual void invertSelection();
QBrush itemTextBackground() const;
ItemTextPos itemTextPos() const;
bool itemsMovable() const;
QIconViewItem *lastItem() const;
int maxItemTextLength() const;
int maxItemWidth() const;
QSize minimumSizeHint() const;
virtual void repaintItem(QIconViewItem *item);
ResizeMode resizeMode() const;
virtual void selectAll(bool select);
SelectionMode selectionMode() const;
virtual void setArrangement(Arrangement am);
virtual void setAutoArrange(bool b);
virtual void setCurrentItem(QIconViewItem *item);
virtual void setFont(const QFont &);
virtual void setGridX(int rx);
virtual void setGridY(int ry);
virtual void setItemTextBackground(const QBrush &b);
virtual void setItemTextPos(ItemTextPos pos);
virtual void setItemsMovable(bool b);
virtual void setMaxItemTextLength(int w);
virtual void setMaxItemWidth(int w);
virtual void setPalette(const QPalette &);
virtual void setResizeMode(ResizeMode am);
virtual void setSelected(QIconViewItem *item, bool s, bool cb = FALSE);
virtual void setSelectionMode(SelectionMode m);
virtual void setShowToolTips(bool b);
void setSorting(bool sort, bool ascending = TRUE);
virtual void setSpacing(int sp);
virtual void setWordWrapIconText(bool b);
virtual void showEvent(QShowEvent *);
```

```
bool showToolTips() const;
QSize sizeHint() const;
QSizePolicy sizePolicy() const;
virtual void sort(bool ascending = TRUE);
bool sortDirection() const;
bool sorting() const;
int spacing() const;
virtual void takeItem(QIconViewItem *item);
bool wordWrapIconText() const;
```

처리부

```
virtual void arrangeItemsInGrid(const QSize &grid, bool update = TRUE);
virtual void arrangeItemsInGrid(bool update = TRUE);
virtual void setContentsPos(int x, int y);
virtual void updateContents();
```

신호

```
void clicked(QIconViewItem *);
void clicked(QIconViewItem *, const QPoint &);
void currentChanged(QIconViewItem *item);
void doubleClicked(QIconViewItem *item);
void dropped(QDropEvent *e, const QValueList < QIconDragItem > &lst);
void itemRenamed(QIconViewItem *item, const QString &);
void itemRenamed(QIconViewItem *item);
void mouseButtonClicked(int button, QIconViewItem *item, const QPoint &pos);
void mouseButtonPressed(int button, QIconViewItem *item, const QPoint &pos);
void moved();
void onItem(QIconViewItem *item);
void onViewport();
void pressed(QIconViewItem *);
void pressed(QIconViewItem *, const QPoint &);
void returnPressed(QIconViewItem *item);
void rightButtonClicked(QIconViewItem *item, const QPoint &pos);
void rightButtonPressed(QIconViewItem *item, const QPoint &pos);
void selectionChanged();
void selectionChanged(QIconViewItem *item);
```

렬거형

```
enum SelectionMode { Single=0, Multi, Extended, NoSelection };
```

```
enum Arrangement { LeftToRight=0, TopToBottom };
enum ResizeMode { Fixed=0, Adjust };
enum ItemTextPos { Bottom=0, Right };
```

다음 실례는 그림 18-7에 보여준 5개 그림기호를 현시한다. 첫째 그림기호는 픽스매프도 본문도 없으므로 기정픽스매프를 사용하고 표식자를 가지지 않는다. 다음 2개 그림기호는 기정픽스매프를 사용하지만 둘다 표식자본문을 가지고있다. 마지막 2개 그림기호는 픽스매프와 표식자를 모두 가지며 "Flag"으로 표식된 그림기호가 마우스에 의해 선택되였다.

```
/* showiconview.cpp */
#include <qapplication.h>
#include <qiconview.h>
int main(int argc,char **argv)
{
    QIconViewItem *item;
    QApplication app(argc,argv);
    QIconView *iconview = new QIconView();
    item = new QIconViewItem(iconview);
    item = new QIconViewItem(iconview, "Icon Label");
    item = new QIconViewItem(iconview, "Icon With\nLong Label");
    QPixmap flag("flag.png");
    item = new QIconViewItem(iconview, "Flag", flag);
    QPixmap idea("idea.png");
    item = new QIconViewItem(iconview, "Idea", idea);
    iconview->show();
    app.setMainWidget(iconview);
    return(app.exec());
}
```
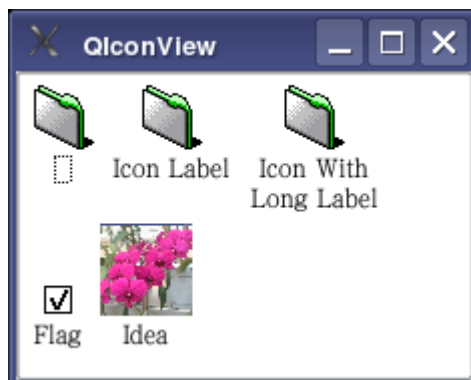
그림 18-7. 5개 그림기호를 현시하는 QIconView창문부품

QInputDialog

QInputDialog창문부품은 정적메쏘드들의 모임이며 매개 메쏘드는 사용자에게 입력을 재촉하는 대화칸을 펼친다.

머리부파일

```
#include <qinputdialog.h>
```

기초클라스

QDialog QObject QPaintDevice QWidget Qt

메쏘드

```
static double getDouble(const QString &caption, const QString &label, double num = 0,
        double from = - 2147483647, double to = 2147483647, int step = 1, bool *ok = 0,
        QWidget *parent = 0, const char *name = 0);
static int getInteger(const QString &caption, const QString &label, int num = 0,
        int from = - 2147483647, int to = 2147483647, int step = 1, bool *ok = 0, QWidget *parent
= 0,
        const char *name = 0);
static QString getItem(const QString &caption, const QString &label, const QStringList &list,
        int current = 0, bool editable = TRUE, bool *ok = 0, QWidget *parent = 0,
        const char *name = 0);
static QString getText(const QString &caption, const QString &label,
        const QString &text = QString::null, bool *ok = 0, QWidget *parent = 0, const char *name =
 0);
```

다음 실례는 1.0-10.0의 double값을 사용자에게 요구한다. 사용자가 OK단추를 선택하면 Boolean값 OK는 true로 설정되고 그렇지 않으면 false로 설정된다. 그림 18-8에 보여주는것처럼 getDouble()에 넘기는 인수는 창문꼭대기에 제목을 설정하고 본문창문우에 재촉문을 현시한다.

```
/* showcolordialog.cpp */
#include <qapplication.h>
#include <qinputdialog.h>
#include <iostream.h>
int main(int argc,char **argv)
{
    bool OK;
    QApplication app(argc,argv);
    double value = QInputDialog::getDouble("A Double Value", "Enter a number from 1.0 to 10.0",
        8.902, 1.0, 10.0, 1, &OK);
    if(OK)
```

```
        cout << "The value is: " << value << endl;
    else
        cout << "No data entered." << endl;
    return(app.exec());
}
```

그림 18-8. double값을 요구하는 QInputDialog대화칸

QLCDNumber

QLCDNumber창문부품은 LCD현시기의 수자들과 같은 서체로 수를 현시한다.

머리부파일

```
#include <qlcdnumber.h>
```

기초클라스

QFrame QObject QPaintDevice QWidget Qt

구성자

```
QLCDNumber(QWidget *parent = 0, const char *name = 0);
QLCDNumber(uint numDigits, QWidget *parent = 0, const char *name = 0);
```

메쏘드

```
bool checkOverflow(double num) const;
bool checkOverflow(int num) const;
int intValue() const;
Mode mode() const;
int numDigits() const;
SegmentStyle segmentStyle() const;
virtual void setMode(Mode);
virtual void setNumDigits(int nDigits);
virtual void setSegmentStyle(SegmentStyle);
QSize sizeHint() const;
QSizePolicy sizePolicy() const;
bool smallDecimalPoint() const;
double value() const;
```

처리부

void display(int num);

void display(double num);

void display(const QString &str);

virtual void setBinMode();

virtual void setDecMode();

virtual void setHexMode();

virtual void setOctMode();

virtual void setSmallDecimalPoint(bool);

신호

void overflow();

렬거형

enum Mode { Hex, HEX=Hex, Dec, DEC=Dec, Oct, OCT=Oct, Bin, BIN=Bin };

enum SegmentStyle { Outline, Filled, Flat };

3장에 QLCDNumber창문부품의 실례가 있다.

QLabel

QLabel창문부품은 임의의 서체로 형식이 없는 본문을 현시한다.

머리부파일

#include <qlabel.h>

기초클라스

QFrame QObject QPaintDevice QWidget Qt

파생클라스

KDockWindow KImageTrackLabel KStatusBarLabel KURLLabel

구성자

QLabel(QWidget *parent, const char *name = 0, WFlags f = 0);

QLabel(const QString &text, QWidget *parent, const char *name = 0, WFlags f = 0);

QLabel(QWidget *buddy, const QString &, QWidget *parent, const char *name = 0, WFlags f = 0);

메쏘드

int alignment() const;

bool autoResize() const;

QWidget *buddy() const;

int heightForWidth(int) const;

int indent() const;

QSize minimumSizeHint() const;

```
QMovie *movie() const;
QPixmap *pixmap() const;
virtual void setAlignment(int);
void setAutoMask(bool);
virtual void setAutoResize(bool);
virtual void setBuddy(QWidget *);
void setIndent(int);
void setTextFormat(TextFormat);
QSize sizeHint() const;
QSizePolicy sizePolicy() const;
QString text() const;
TextFormat textFormat() const;
```

처리부

```
void clear();
virtual void setMovie(const QMovie &);
virtual void setNum(int);
virtual void setNum(double);
virtual void setPixmap(const QPixmap &);
virtual void setText(const QString &);
```

QLineEdit

QLineEdit창문부품은 주로 자료입력에 쓰이는 간단한 한행본문편집기이다.

머리부파일

```
#include <qlineedit.h>
```

기초클라스

QObject QPaintDevice QWidget Qt

파생클라스

KAccelInput KDateInternalYearSelector KLineEdit KPasswordEdit KRestrictedLine

구성자

```
QLineEdit(QWidget *parent, const char *name = 0);
QLineEdit(const QString &, QWidget *parent, const char *name = 0);
```

메쏘드

```
int alignment() const;
void backspace();
void copy() const;
```

```
void cursorLeft(bool mark, int steps = 1);
int cursorPosition() const;
void cursorRight(bool mark, int steps = 1);
void cursorWordBackward(bool mark);
void cursorWordForward(bool mark);
void cut();
void del();
QString displayText() const;
EchoMode echoMode() const;
bool edited() const;
void end(bool mark);
bool frame() const;
bool hasMarkedText() const;
void home(bool mark);
bool isReadOnly() const;
QString markedText() const;
int maxLength() const;
QSize minimumSizeHint() const;
void paste();
void setAlignment(int flag);
virtual void setCursorPosition(int);
virtual void setEchoMode(EchoMode);
void setEdited(bool);
virtual void setEnabled(bool);
virtual void setFont(const QFont &);
virtual void setFrame(bool);
virtual void setMaxLength(int);
virtual void setPalette(const QPalette &);
void setReadOnly(bool);
virtual void setSelection(int, int);
virtual void setValidator(const QValidator *);
QSize sizeHint() const;
QSizePolicy sizePolicy() const;
QString text() const;
bool validateAndSet(const QString &, int, int, int);
const QValidator *validator() const;
```

처리부

```
void clear();
void clearValidator();
void deselect();
void insert(const QString &);
void selectAll();
virtual void setText(const QString &);
```

신호

```
void returnPressed();
void textChanged(const QString &);
```

렬거형

```
enum EchoMode { Normal, NoEcho, Password };
```

4장에 QLineEdit창문부품을 사용하는 실례가 있다

QListBox

QListBox창문부품은 마우스로 선택할수 있는 항목들의 목록을 현시한다.

머리부파일

```
#include <qlistbox.h>
```

기초클라스

QFrame QObject QPaintDevice QScrollView QWidget Qt

파생클라스

KListBox KSplitList

구성자

```
QListBox(QWidget *parent = 0, const char *name = 0, WFlags f = 0);
```

메쏘드

```
bool autoBottomScrollBar() const;
bool autoScroll() const;
bool autoScrollBar() const;
bool autoUpdate() const;
bool bottomScrollBar() const;
int cellHeight(int i) const;
int cellHeight() const;
int cellWidth() const;
int cellWidth(int i) const;
void centerCurrentItem();
```

```
void changeItem(const QListBoxItem *, int index);
void changeItem(const QString &text, int index);
void changeItem(const QPixmap &pixmap, int index);
void changeItem(const QPixmap &pixmap, const QString &text, int index);
void clear();
LayoutMode columnMode() const;
uint count() const;
int currentItem() const;
QString currentText() const;
bool dragSelect() const;
QListBoxItem *findItem(const QString &text) const;
QListBoxItem *firstItem() const;
void inSort(const QListBoxItem *);
void inSort(const QString &text);
int index(const QListBoxItem *) const;
void insertItem(const QListBoxItem *, int index = - 1);
void insertItem(const QListBoxItem *,
const QListBoxItem *after);
void insertItem(const QString &text, int index = - 1);
void insertItem(const QPixmap &pixmap, int index = - 1);
void insertItem(const QPixmap &pixmap, const QString &text, int index = - 1);
void insertStrList(const QStrList *, int index = - 1);
void insertStrList(const QStrList &, int index = - 1);
void insertStrList(const char **, int numStrings = - 1, int index = - 1);
void insertStringList(const QStringList &, int index = - 1);
bool isMultiSelection() const;
bool isSelected(int) const;
bool isSelected(const QListBoxItem *) const;
QListBoxItem *item(int index) const;
QListBoxItem *itemAt(QPoint) const;
int itemHeight(int index = 0) const;
QRect itemRect(QListBoxItem *item) const;
bool itemVisible(int index);
bool itemVisible(const QListBoxItem *);
long maxItemWidth() const;
QSize minimumSizeHint() const;
```

```
int numCols() const;
int numColumns() const;
int numItemsVisible() const;
int numRows() const;
const QPixmap *pixmap(int index) const;
void removeItem(int index);
LayoutMode rowMode() const;
bool scrollBar() const;
SelectionMode selectionMode() const;
void setAutoBottomScrollBar(bool enable);
void setAutoScroll(bool);
void setAutoScrollBar(bool enable);
void setAutoUpdate(bool);
virtual void setBottomItem(int index);
void setBottomScrollBar(bool enable);
virtual void setColumnMode(LayoutMode);
virtual void setColumnMode(int);
virtual void setCurrentItem(int index);
virtual void setCurrentItem(QListBoxItem *);
void setDragSelect(bool);
void setFixedVisibleLines(int lines);
virtual void setFont(const QFont &);
void setMultiSelection(bool multi);
virtual void setRowMode(LayoutMode);
virtual void setRowMode(int);
void setScrollBar(bool enable);
virtual void setSelected(QListBoxItem *, bool);
void setSelected(int, bool);
virtual void setSelectionMode(SelectionMode);
void setSmoothScrolling(bool);
virtual void setTopItem(int index);
virtual void setVariableHeight(bool);
virtual void setVariableWidth(bool);
QSize sizeHint() const;
bool smoothScrolling() const;
void sort(bool ascending = TRUE);
```

```
void takeItem(const QListBoxItem *);
QString text(int index) const;
int topItem() const;
void triggerUpdate(bool doLayout);
bool variableHeight() const;
bool variableWidth() const;
void viewportPaintEvent(QPaintEvent *);
```

처리부

```
virtual void clearSelection();
virtual void ensureCurrentVisible();
void invertSelection();
void selectAll(bool select);
```

신호

```
void clicked(QListBoxItem *);
void clicked(QListBoxItem *, const QPoint &);
void currentChanged(QListBoxItem *);
void doubleClicked(QListBoxItem *);
void highlighted(int index);
void highlighted(const QString &);
void highlighted(QListBoxItem *);
void mouseButtonClicked(int, QListBoxItem *, const QPoint &);
void mouseButtonPressed(int, QListBoxItem *, const QPoint &);
void onItem(QListBoxItem *item);
void onViewport();
void pressed(QListBoxItem *);
void pressed(QListBoxItem *, const QPoint &);
void returnPressed(QListBoxItem *);
void rightButtonClicked(QListBoxItem *, const QPoint &);
void rightButtonPressed(QListBoxItem *, const QPoint &);
void selected(int index);
void selected(const QString &);
void selected(QListBoxItem *);
void selectionChanged();
void selectionChanged(QListBoxItem *);
```

렬거형

```
enum SelectionMode { Single, Multi, Extended, NoSelection };
```

```
enum LayoutMode { FixedNumber, FitToWidth, FitToHeight=FitToWidth, Variable };
```

3장과 8장에 QListBox창문부품의 실례가 있다.

QListView

QListView창문부품은 나무구조의 항목목록을 현시하고 마우스를 리용하여 매 하옥들을 열람할수 있게 한다.

머리부파일

```
#include <qlistview.h>
```

기초클라스

QFrame QObject QPaintDevice QScrollView QWidget Qt

파생클라스

KApplicationTree KFileDetailView KListView

구성자

```
QListView(QWidget *parent = 0, const char *name = 0);
```

메쏘드

```
virtual int addColumn(const QString &label, int size = - 1);
virtual int addColumn(const QIconSet &iconset, const QString &label, int size = - 1);
bool allColumnsShowFocus() const;
int childCount() const;
virtual void clear();
virtual void clearSelection();
int columnAlignment(int) const;
QString columnText(int column) const;
int columnWidth(int column) const;
WidthMode columnWidthMode(int column) const;
int columns() const;
QListViewItem *currentItem() const;
void ensureItemVisible(const QListViewItem *);
bool eventFilter(QObject *o, QEvent *);
QListViewItem *firstChild() const;
QHeader *header() const;
virtual void insertItem(QListViewItem *);
bool isMultiSelection() const;
bool isOpen(const QListViewItem *) const;
bool isSelected(const QListViewItem *) const;
```

```
QListViewItem *itemAt(const QPoint &screenPos) const;
int itemMargin() const;
int itemPos(const QListViewItem *);
QRect itemRect(const QListViewItem *) const;
QSize minimumSizeHint() const;
void removeColumn(int index);
virtual void removeItem(QListViewItem *);
void repaintItem(const QListViewItem *) const;
bool rootIsDecorated() const;
QListViewItem *selectedItem() const;
SelectionMode selectionMode() const;
virtual void setAllColumnsShowFocus(bool);
virtual void setColumnAlignment(int, int);
virtual void setColumnText(int column, const QString &label);
virtual void setColumnText(int column, const QIconSet &iconset, const QString &label);
virtual void setColumnWidth(int column, int width);
virtual void setColumnWidthMode(int column, WidthMode);
virtual void setCurrentItem(QListViewItem *);
virtual void setFont(const QFont &);
virtual void setItemMargin(int);
virtual void setMultiSelection(bool enable);
virtual void setOpen(QListViewItem *, bool);
virtual void setPalette(const QPalette &);
virtual void setRootIsDecorated(bool);
virtual void setSelected(QListViewItem *, bool);
void setSelectionMode(SelectionMode mode);
void setShowSortIndicator(bool show);
virtual void setSorting(int column, bool increasing = TRUE);
virtual void setTreeStepSize(int);
void show();
bool showSortIndicator() const;
QSize sizeHint() const;
void sort();
virtual void takeItem(QListViewItem *);
int treeStepSize() const;
```

처리부

```
void invertSelection();
void selectAll(bool select);
void setContentsPos(int x, int y);
void triggerUpdate();
```

신호

```
void clicked(QListViewItem *);
void clicked(QListViewItem *, const QPoint &, int);
void collapsed(QListViewItem *item);
void currentChanged(QListViewItem *);
void doubleClicked(QListViewItem *);
void expanded(QListViewItem *item);
void mouseButtonClicked(int, QListViewItem *, const QPoint &, int);
void mouseButtonPressed(int, QListViewItem *, const QPoint &, int);
void onItem(QListViewItem *item);
void onViewport();
void pressed(QListViewItem *);
void pressed(QListViewItem *, const QPoint &, int);
void returnPressed(QListViewItem *);
void rightButtonClicked(QListViewItem *, const QPoint &, int);
void rightButtonPressed(QListViewItem *, const QPoint &, int);
void selectionChanged();
void selectionChanged(QListViewItem *);
```

렬거형

```
enum WidthMode { Manual, Maximum };
enum SelectionMode { Single, Multi, Extended, NoSelection };
```

QListView창문부품에 현시된 나무들은 여러준위이고 보조나무들을 마우스로 펼치고 닫을수 있다. 나무들은 여러 층의 깊이를 가질수 있고 나무의 마디는 많은 창문부품들중의 하나일수 있다. 그러므로 각이한 렬들을 만들수 있고 많은 나무들이 있을수 있다. 흘림띠는 바닥에 나타나고 필요하다면 오른쪽에도 나타나며 여러 렬들이 사용되면 매개 나무는 렬제목들의 크기를 변경하여 크기를 조종할수 있다.

다음 실례는 검사칸에 의해서 표시된 잎매듭들을 가지는 하나의 나무를 만든다. 매개 내부나무마디(자식마디들을 조종할 능력이 있는 마디)는 QListViewItem객체이다. QValueList형타는 QListViewItem객체들의 모임을 담고있는 목록을 만드는데 사용된다. QValueList의 append()는 매개 QListViewItem객체를 삽입하는데 사용된다. 내부순환고리는 5개의

QCheckListItem객체를 창조하고 매개 객체를 표시하는 마디열에는 QCheckBox가 있다. 그림 18-9에서 결과나무를 보여준다. 부모마디들중 2개는 열리고 검사칸 3개가 선택된 상태로 되여있다.

```
/* showlistiew.cpp */
#include <qapplication.h>
#include <qlistview.h>
int main(int argc,char **argv)
{
    QApplication app(argc,argv);
    QListView *listview = new QListView(0);
    listview->show();
    listview->addColumn("Column Heading");
    listview->setRootIsDecorated(TRUE);
    QValueList<QListViewItem *> valuelist;
    for(int i=1; i<6; i++) {
        QListViewItem *viewitem = new QListViewItem(listview, QString("Parent %1").arg(i));
        valuelist.append(viewitem);
        for(int j=1; j<6; j++) {
            new QCheckListItem(viewitem, QString("Child %1 of Parent %2").arg(j).arg(i),
                QCheckListItem::CheckBox);
        }
    }
    listview->show();
    app.setMainWidget(listview);
    return(app.exec());
}
```
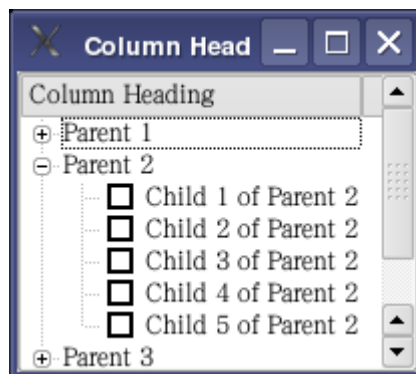
그림 18-9. 5개 부모마디를 가지는 QListView

QMainWindow

QMainWindow는 보통 응용프로그람의 기본창문이 요구하는 기능을 제공하는 제일웃준위창문으로 사용된다. 그것은 차림표띠, 도구띠 그리고 상태띠를 제공한다.

머리부파일

```
#include <qmainwindow.h>
```

기초클라스

QObject QPaintDevice QWidget Qt

구성자

```
QMainWindow(QWidget *parent = 0, const char *name = 0, WFlags f = WType_TopLevel);
```

메쏘드

```
void addToolBar(QToolBar *, ToolBarDock = Top, bool newLine = FALSE);
void addToolBar(QToolBar *, const QString &label, ToolBarDock = Top, bool newLine = FALSE);
QWidget *centralWidget() const;
bool eventFilter(QObject *, QEvent *);
bool getLocation(QToolBar *tb, ToolBarDock &dock, int &index, bool &nl, int &extraOffset) const;
bool isDockEnabled(ToolBarDock dock) const;
bool isDockEnabled(QToolBar *tb, ToolBarDock dock) const;
bool isDockMenuEnabled() const;
void lineUpToolBars(bool keepNewLines = FALSE);
QMenuBar *menuBar() const;
QSize minimumSizeHint() const;
void moveToolBar(QToolBar *, ToolBarDock = Top);
void moveToolBar(QToolBar *, ToolBarDock, bool nl, int index, int extraOffset = - 1);
bool opaqueMoving() const;
void removeToolBar(QToolBar *);
bool rightJustification() const;
virtual void setCentralWidget(QWidget *);
virtual void setDockEnabled(ToolBarDock dock, bool enable);
void setDockEnabled(QToolBar *tb, ToolBarDock dock, bool enable);
void show();
QSize sizeHint() const;
QStatusBar *statusBar() const;
QList<QToolBar> toolBars(ToolBarDock dock) const;
bool toolBarsMovable() const;
QToolTipGroup *toolTipGroup() const;
```

```
bool usesBigPixmaps() const;
bool usesTextLabel() const;
```

처리부

```
void setDockMenuEnabled(bool);
void setOpaqueMoving(bool);
virtual void setRightJustification(bool);
void setToolBarsMovable(bool);
virtual void setUsesBigPixmaps(bool);
void setUsesTextLabel(bool);
void whatsThis();
```

신호

```
void endMovingToolBar(QToolBar *);
void pixmapSizeChanged(bool);
void startMovingToolBar(QToolBar *);
void toolBarPositionChanged(QToolBar *);
void usesTextLabelChanged(bool);
```

렬거형

```
enum ToolBarDock { Unmanaged, TornOff, Top, Bottom, Right, Left, Minimized };
```

다음 실례는 응용프로그람의 제일웃준위창문으로서 QMainWindow창문부품을 설정하는 방법을 보여준다. QMainWindow창문부품은 다른 구성요소들(QMainWindow의 메쏘드들을 통하여 접근할수 있는것)의 용기로 리용된다. 이 구성요소들은 QMenuBar, QStatusBar, QToolTipGroup 그리고 QToolBar객체들의 목록이다. 또한 중심에 QWidget를 포함하는데 그것은 응용프로그람의 기본현시창문으로 된다.

```
/* showmainwindow.cpp */
#include <qapplication.h>
#include <qmainwindow.h>
int main(int argc,char **argv)
{
    QApplication app(argc,argv);
    QMainWindow *mainwindow = new QMainWindow();
    mainwindow->show();
    app.setMainWidget(mainwindow);
    return(app.exec());
}
```

QMenuBar

QMenuBar는 튀여나오기차림표그룹사이의 관계를 관리하는 수평띠이다.

머리부파일

```
#include <qmenubar.h>
```

기초클라스

QFrame QMenuData QObject QPaintDevice QWidget Qt

파생클라스

KMenuBar

구성자

```
QMenuBar(QWidget *parent = 0, const char *name = 0);
```

메쏘드

```
bool customWhatsThis() const;
bool eventFilter(QObject *, QEvent *);
int heightForWidth(int) const;
void hide();
bool isDefaultUp() const;
QSize minimumSize() const;
QSize minimumSizeHint() const;
Separator separator() const;
void setDefaultUp(bool);
virtual void setSeparator(Separator when);
void show();
QSize sizeHint() const;
void updateItem(int id);
```

신호

```
void activated(int itemId);
void highlighted(int itemId);
```

렬거형

```
enum Separator { Never=0, InWindowsStyle=1 };
```

다음 실례는 2개의 튀여나오기차림표를 포함하는 QMenuBar를 보여준다. 매개 튀여나오기차림표는 하나의 차림표항목을 포함한다. 그림 18-10은 두번째 튀여나오기차림표가 능동으로 된 차림표띠를 보여준다.

```
/* showmenubar.cpp */
#include <qapplication.h>
#include <qmenubar.h>
```

```
int main(int argc,char **argv)
{
    QApplication app(argc,argv);
    QMenuBar *menubar = new QMenuBar();
    menubar->setSeparator(QMenuBar::InWindowsStyle);
    QPopupMenu* filePopup = new QPopupMenu();
    filePopup->insertItem("&Quit", &app, SLOT(quit()));
    menubar->insertItem("&File", filePopup);
    QPopupMenu* editPopup = new QPopupMenu();
    editPopup->insertItem("&Paste");
    menubar->insertItem("E&dit",editPopup);
    menubar->show();
    app.setMainWidget(menubar);
    return(app.exec());
}
```
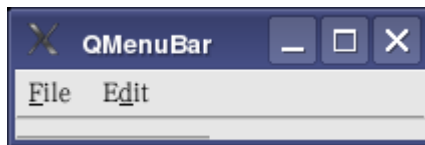
그림 18-10. 2개의 튀여나오기차림표를 가진 QMenuBar

QMessageBox

QMessageBox는 사용자에게 정보를 현시하기 위하여 펼치고 응답을 기다리는 대화칸이다. 그것은 여러가지 그림기호들과 선택단추들을 비롯한 많은 요소가 있다.

머리부파일

```
#include <qmessagebox.h>
```

기초클라스

QDialog QObject QPaintDevice QWidget Qt

구성자

```
QMessageBox(QWidget *parent = 0, const char *name = 0);
QMessageBox(const QString &caption, const QString &text, Icon icon, int button0, int button1, int button2,
QWidget *parent = 0, const char *name = 0, bool modal = TRUE, WFlags f = WStyle_DialogBorder);
```

메쏘드

```
static void about(QWidget *parent, const QString &caption, const QString &text);
```

```
static void aboutQt(QWidget *parent, const QString &caption = QString::null);
void adjustSize();
QString buttonText(int button) const;
static int critical(QWidget *parent, const QString &caption, const QString &text, int button0,
        int button1, int button2 = 0);
static int critical(QWidget *parent, const QString &caption, const QString &text,
        const QString &button0Text = QString::null, const QString &button1Text = QString::null,
        const QString &button2Text = QString::null, int defaultButtonNumber = 0,
        int escapeButtonNumber = - 1);
Icon icon() const;
const QPixmap *iconPixmap() const;
static int information(QWidget *parent, const QString &caption, const QString &text, int button0,
        int button1 = 0, int button2 = 0);
static int information(QWidget *parent, const QString &caption, const QString &text,
        const QString &button0Text = QString::null, const QString &button1Text = QString::null,
        const QString &button2Text = QString::null, int defaultButtonNumber = 0,
        int escapeButtonNumber = - 1);
static int message(const QString &caption, const QString &text,
        const QString &buttonText = QString::null, QWidget *parent = 0, const char *name = 0);
static bool query(const QString &caption, const QString &text,
        const QString &yesButtonText = QString::null, const QString &noButtonText = QString::null,
        QWidget *parent = 0, const char *name = 0);
void setButtonText(int button, const QString &);
void setIcon(Icon);
void setIcon(const QPixmap &);
void setIconPixmap(const QPixmap &);
void setText(const QString &);
void setTextFormat(TextFormat);
static QPixmap standardIcon(Icon icon, GUIStyle style);
QString text() const;
TextFormat textFormat() const;
static int warning(QWidget *parent, const QString &caption, const QString &text, int button0,
        int button1, int button2 = 0);
static int warning(QWidget *parent, const QString &caption, const QString &text,
        const QString &button0Text = QString::null, const QString &button1Text = QString::null,
        const QString &button2Text = QString::null, int defaultButtonNumber = 0,
```

```
int escapeButtonNumber = - 1);
```

렬거형

```
enum Icon { NoIcon=0, Information=1, Warning=2, Critical=3 };
enum (anon) { Ok=1, Cancel=2, Yes=3, No=4, Abort=5, Retry=6, Ignore=7, ButtonMask=0x07,
    Default=0x100, Escape=0x200, FlagMask=0x300 };
```

다음 실례는 정적메쏘드들중 하나를 사용하여 미리 구성된 QMessageBox를 펼친다. 그림 18-11은 정보그림기호를 가지는 통보칸을 보여준다.

```
/* showmessgebox.cpp */
#include <qapplication.h>
#include <qmessagebox.h>
int main(int argc,char **argv)
{
    QApplication app(argc,argv);
    QMessageBox::information(0, "The Caption of an Informaton Box",
        "This is a QMessageBox that is configured\n"
        "to display information to the user and\n"
        "wait for a response.");
    return(app.exec());
}
```
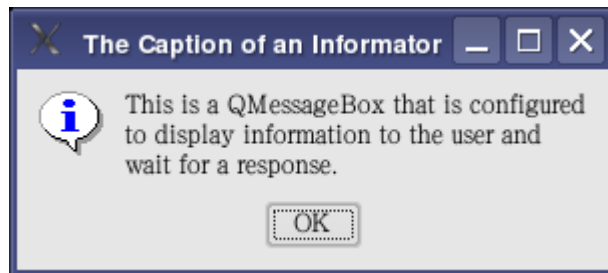
그림 18-11. 정보그림기호를 사용하는 QMessageBox

QMultiLineEdit

QMultiLineEdit창문부품은 사용자가 본문을 입력하고 현존본문을 수정할수 있는 문서편집기이다.

머리부파일

```
#include <qmultilineedit.h>
```

기초클라스

QFrame QObject QPaintDevice QTableView QWidget Qt

파생클라스

KEdit

구성자

```
QMultiLineEdit(QWidget *parent = 0, const char *name = 0);
```

메쏘드

```
int alignment() const;
bool atBeginning() const;
bool atEnd() const;
bool autoUpdate() const;
void cursorPosition(int *line, int *col) const;
void cursorWordBackward(bool mark);
void cursorWordForward(bool mark);
static int defaultTabStop();
EchoMode echoMode() const;
bool edited() const;
void getCursorPosition(int *line, int *col) const;
int hMargin() const;
virtual void insertAt(const QString &s, int line, int col, bool mark = FALSE);
virtual void insertLine(const QString &s, int line = - 1);
bool isOverwriteMode() const;
bool isReadOnly() const;
bool isUndoEnabled() const;
int length() const;
int maxLength() const;
int maxLineLength() const;
int maxLineWidth() const;
int maxLines() const;
QSize minimumSizeHint() const;
int numLines() const;
virtual void removeLine(int line);
void setAlignment(int flags);
virtual void setAutoUpdate(bool);
virtual void setCursorPosition(int line, int col, bool mark = FALSE);
static void setDefaultTabStop(int ex);
virtual void setEchoMode(EchoMode);
void setEdited(bool);
virtual void setFixedVisibleLines(int lines);
```

```
virtual void setFont(const QFont &font);
virtual void setHMargin(int);
void setMaxLength(int);
virtual void setMaxLineLength(int);
virtual void setMaxLines(int);
virtual void setSelection(int row_from, int col_from, int row_to, int col_t);
void setUndoDepth(int);
void setUndoEnabled(bool);
virtual void setValidator(const QValidator *);
void setWordWrap(WordWrap mode);
void setWrapColumnOrWidth(int);
void setWrapPolicy(WrapPolicy policy);
QSize sizeHint() const;
QSizePolicy sizePolicy() const;
QString text() const;
QString textLine(int line) const;
int undoDepth() const;
const QValidator *validator() const;
WordWrap wordWrap() const;
int wrapColumnOrWidth() const;
WrapPolicy wrapPolicy() const;
```

처리부

```
void append(const QString &);
void clear();
void copy() const;
void copyText() const;
void cut();
void deselect();
void insert(const QString &);
void paste();
void redo();
void selectAll();
virtual void setOverwriteMode(bool);
virtual void setReadOnly(bool);
virtual void setText(const QString &);
void undo();
```

신호

```
void redoAvailable(bool);
void returnPressed();
void textChanged();
void undoAvailable(bool);
```

렬거형

```
enum EchoMode { Normal, NoEcho, Password };
enum WordWrap { NoWrap, WidgetWidth, FixedPixelWidth, FixedColumnWidth };
enum WrapPolicy { AtWhiteSpace, Anywhere };
```

8장에 QMultiLineEdit창문부품을 사용하는 실례들이 있다.

QPopupMenu

QPopupMenu창문부품은 튀여나오기차림표이다. 그것은 보통 차림표띠 혹은 부모튀여나오기차림표의 성원으로 나타난다.

머리부파일

```
#include <qpopupmenu.h>
```

기초클라스

QFrame QMenuData QObject QPaintDevice QWidget Qt

파생클라스

KAccelMenu KPopupMenu KPopupMenu

구성자

```
QPopupMenu(QWidget *parent = 0, const char *name = 0);
```

메쏘드

```
bool customWhatsThis() const;
int exec();
int exec(const QPoint &pos, int indexAtPoint = 0);
void hide();
int idAt(int index) const;
int idAt(const QPoint &pos) const;
int insertTearOffHandle(int id = - 1, int index = - 1);
bool isCheckable() const;
void popup(const QPoint &pos, int indexAtPoint = 0);
virtual void setActiveItem(int);
virtual void setCheckable(bool);
void setFont(const QFont &);
```

```
void show();
QSize sizeHint() const;
void updateItem(int id);
```

신호

```
void aboutToShow();
void activated(int itemId);
void activatedRedirect(int itemId);
void highlighted(int itemId);
void highlightedRedirect(int itemId);
```

6장에 QPopupMenu의 실례들이 있다

QPrintDialog

QprintDialog창문부품은 사용자가 응용프로그람에 의해 인쇄환경을 구성하고 인쇄를 조종할수 있는 대면부이다.

머리부파일

```
#include <qprintdialog.h>
```

기초클라스

QDialog QObject QPaintDevice QWidget Qt

구성자

```
QPrintDialog(QPrinter *, QWidget *parent = 0, const char *name = 0);
```

메쏘드

```
void addButton(QPushButton *but);
static bool getPrinterSetup(QPrinter *);
QPrinter *printer() const;
void setPrinter(QPrinter *, bool = FALSE);
```

QPrintDialog의 다음 실례는 그림 18-12에 보여주는 대화칸창문을 펼치는데 QPrinter객체를 사용한다.

```
/* showprintdialog.cpp */
#include <qapplication.h>
#include <qprintdialog.h>
#include <qprinter.h>
#include <iostream.h>
int main(int argc,char **argv)
{
   QApplication app(argc,argv);
```

```
    QPrinter *printer = new QPrinter();
    bool OK = QPrintDialog::getPrinterSetup(printer);
    if(OK)
        cout << "Printer configuration set." << endl;
    else
        cout << "Printer configuration not set." << endl;
    return(app.exec());
}
```
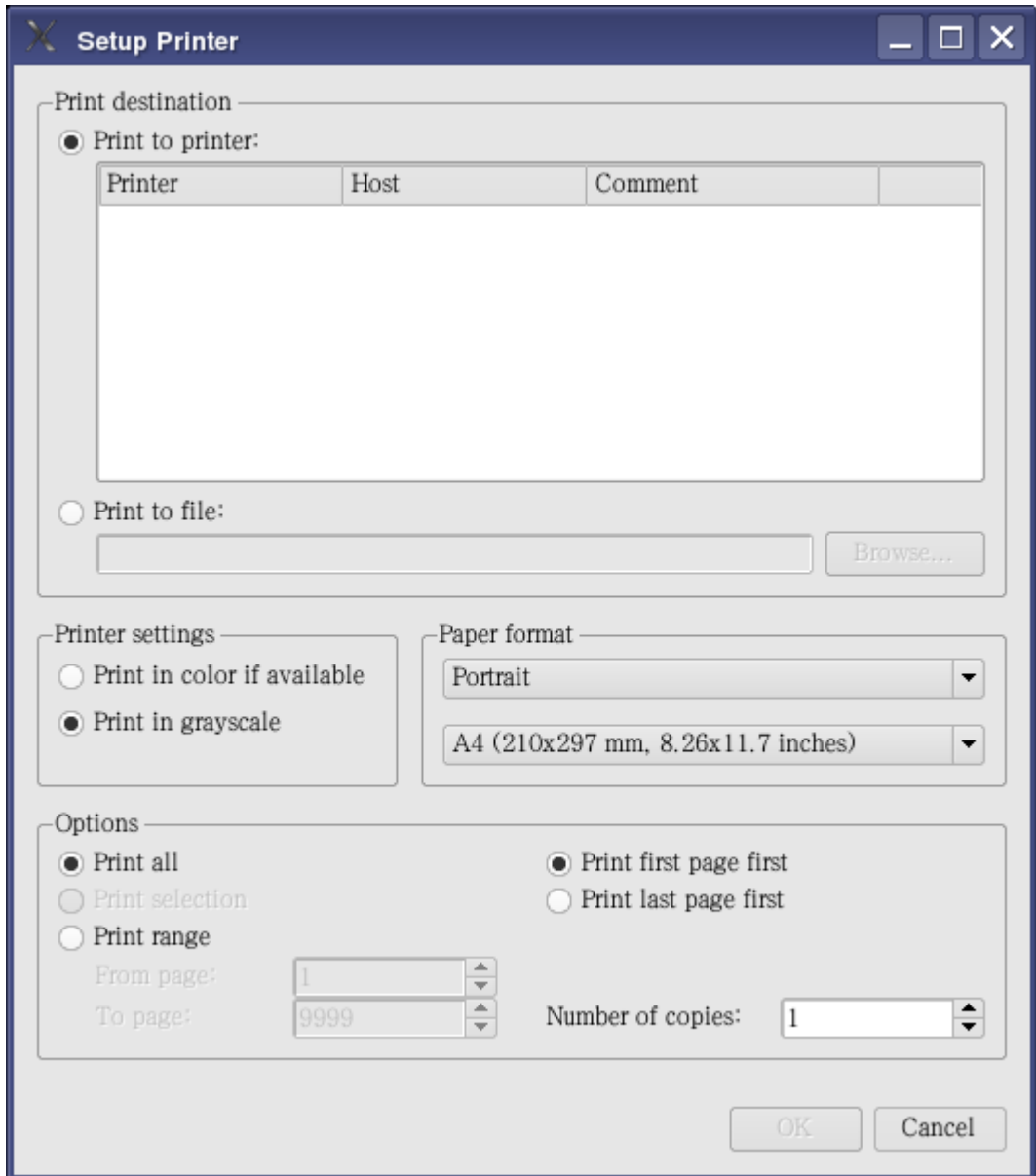
그림 18-12. QPrintDialog창문

QProgressBar

QProgressBar는 수평진척상황띠이다.

머리부파일

```
#include <qprogressbar.h>
```

기초클라스

QFrame QObject QPaintDevice QWidget Qt

구성자

```
QProgressBar(QWidget *parent = 0, const char *name = 0, WFlags f = 0);
QProgressBar(int totalSteps, QWidget *parent = 0, const char *name = 0, WFlags f = 0);
```

메쏘드

```
bool centerIndicator() const;
bool indicatorFollowsStyle() const;
QSize minimumSizeHint() const;
int progress() const;
void setCenterIndicator(bool on);
void setIndicatorFollowsStyle(bool);
void show();
QSize sizeHint() const;
QSizePolicy sizePolicy() const;
int totalSteps() const;
```

처리부

```
void reset();
virtual void setProgress(int progress);
virtual void setTotalSteps(int totalSteps);
```

다음 실례는 그림 18-13에 보여주는것처럼 총 200걸음에서 현재 122걸음에 왔다는 상황띠를 보여준다. 그것은 61%의 걸음과 등가하다.

```
/* showprogressbar.cpp */
#include <qapplication.h>
#include <qprogressbar.h>
int main(int argc,char **argv)
{
   QApplication app(argc,argv);
   QProgressBar *progressbar = new QProgressBar();
   progressbar->setTotalSteps(200);
   progressbar->setProgress(122);
```

```
    progressbar->show();
    app.setMainWidget(progressbar);
    return(app.exec());
}
```

그림 18-13. 실행결음을 보여주는 QProgressBar

QProgressDialog

QProgressBar는 Cancel단추를 가진 수평진척상황띠이다. 이것은 QDialog로부터 계승되는 튀여나오기창문이 아니라 창문부품이다.

머리부파일

#include <qprogressdialog.h>

기초클라스

QObject QPaintDevice QSemiModal QWidget Qt

구성자

QProgressDialog(QWidget *parent = 0, const char *name = 0, bool modal = FALSE, WFlags f = 0);

QProgressDialog(const QString &labelText, const QString &cancelButtonText, int totalSteps,
    QWidget *parent = 0, const char *name = 0, bool modal = FALSE, WFlags f = 0);

메쏘드

bool autoClose() const;

bool autoReset() const;

QString labelText() const;

int minimumDuration() const;

int progress() const;

void setAutoClose(bool b);

void setAutoReset(bool b);

void setBar(QProgressBar *);

void setCancelButton(QPushButton *);

void setLabel(QLabel *);

QSize sizeHint() const;

int totalSteps() const;

bool wasCancelled() const;

처리부

```
void cancel();
void reset();
void setCancelButtonText(const QString &);
void setLabelText(const QString &);
void setMinimumDuration(int ms);
void setProgress(int progress);
void setTotalSteps(int totalSteps);
```

신호

```
void cancelled();
```

다음 실례는 그림 18-14에 보여주는것처럼 총 200걸음을 가지는데 현재 122걸음의 진척상황띠를 보여준다. Cancel단추는 cancelled()신호를 발생하고 창문을 닫는다.

```
/* showprogressdialog.cpp */
#include <qapplication.h>
#include <qprogressdialog.h>
int main(int argc,char **argv)
{
   QApplication app(argc,argv);
   QProgressDialog *progressdialog = new QProgressDialog();
   progressdialog->setTotalSteps(200);
   progressdialog->setProgress(122);
   progressdialog->show();
   app.setMainWidget(progressdialog);
   return(app.exec());
}
```
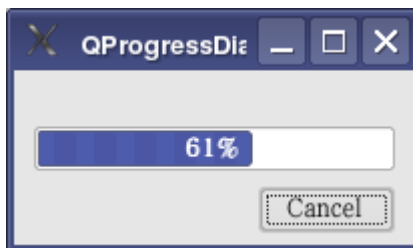
그림 18-14. 진행상황과 Cancel단추를 보여주는 QProgressDialog

QPushButton

QPushButton은 그 형태를 바꾸어 마우스에 응답하는 경사진 경계를 가지는 창문부품이다.

머리부파일

```
#include <qpushbutton.h>
```

기초클라스

QButton QObject QPaintDevice QWidget Qt

파생클라스

KColorButton KDialogBaseButton KDockButton_Private KIconButton KKeyButton

구성자

```
QPushButton(QWidget *parent, const char *name = 0);
QPushButton(const QString &text, QWidget *parent, const char *name = 0);
QPushButton(const QIconSet &icon, const QString &text, QWidget *parent, const char *name = 0);
```

메쏘드

```
bool autoDefault() const;
QIconSet *iconSet() const;
bool isDefault() const;
bool isMenuButton() const;
void move(int x, int y);
void move(const QPoint &p);
QPopupMenu *popup() const;
void resize(int w, int h);
void resize(const QSize &);
virtual void setAutoDefault(bool autoDef);
virtual void setDefault(bool def);
virtual void setGeometry(int x, int y, int w, int h);
virtual void setGeometry(const QRect &);
void setIconSet(const QIconSet &);
virtual void setIsMenuButton(bool);
void setPopup(QPopupMenu *popup);
virtual void setToggleButton(bool);
QSize sizeHint() const;
QSizePolicy sizePolicy() const;
```

처리부

```
virtual void setOn(bool);
void toggle();
```

QRadioButton

QRadioButton은 마우스에 의해 on 또는 off로 절환될수 있는 단추이다. 다른 라디오단추들을 가지는 그룹에 포함될 때에는 오직 그것들중의 하나가 임의의 시각에 on으로 될수 있다.

머리부파일

```
#include <qradiobutton.h>
```

기초클라스

QButton QObject QPaintDevice QWidget Qt

구성자

```
QRadioButton(QWidget *parent, const char *name = 0);
QRadioButton(const QString &text, QWidget *parent, const char *name = 0);
```

메쏘드

```
bool isChecked() const;
virtual void setChecked(bool check);
QSize sizeHint() const;
QSizePolicy sizePolicy() const;
```

7장에 QRadioButton의 많은 실례가 있다.

QScrollBar

QScrollBar창문부품은 수직 또는 수평흘림띠로서 구성될수 있다.

머리부파일

```
#include <qscrollbar.h>
```

기초클라스

QObject QPaintDevice QRangeControl QWidget Qt

구성자

```
QScrollBar(QWidget *parent, const char *name = 0);
QScrollBar(Orientation, QWidget *parent, const char *name = 0);
QScrollBar(int minValue, int maxValue, int LineStep, int PageStep, int value, Orientation,
        QWidget *parent, const char *name = 0);
```

메쏘드

```
bool draggingSlider() const;
int lineStep() const;
int maxValue() const;
int minValue() const;
```

```
Orientation orientation() const;
int pageStep() const;
void setLineStep(int);
void setMaxValue(int);
void setMinValue(int);
virtual void setOrientation(Orientation);
void setPageStep(int);
virtual void setPalette(const QPalette &);
virtual void setTracking(bool enable);
void setValue(int);
QSize sizeHint() const;
QSizePolicy sizePolicy() const;
bool tracking() const;
int value() const;
```

신호

```
void nextLine();
void nextPage();
void prevLine();
void prevPage();
void sliderMoved(int value);
void sliderPressed();
void sliderReleased();
void valueChanged(int value);
```

대부분의 창문부품들과 달리 QScrollBar창문부품의 모든 구성자는 부모창문부품을 요구한다. 다음 실례는 2개의 QScrollBar객체를 만든다. 하나는 수평으로, 다른 하나는 수직으로 향한다. 그림 18-15에 보여준것처럼 부모창문부품과 2개 흘림띠의 기하학적배치는 흘림띠들이 바닥의 표준위치와 창문의 오른쪽에 나타나도록 고정된다.

```
/* showscrollbar.cpp */
#include <qapplication.h>
#include <qscrollbar.h>
int main(int argc,char **argv)
{
    QApplication app(argc,argv);
    QWidget *widget = new QWidget();
    QScrollBar *vscrollbar = new QScrollBar(Qt::Vertical,widget);
    vscrollbar->setGeometry(200,0,30,200);
```

```
    QScrollBar *hscrollbar = new QScrollBar(Qt::Horizontal,widget);
    hscrollbar->setGeometry(0,200,200,30);
    widget->setFixedSize(230,230);
    widget->show();
    app.setMainWidget(widget);
    return(app.exec());
}
```

그림18 15. 수평 및 수직QScrollBar

QScrollView

QScrollView창문부품은 하나의 자식창문부품을 보유하는 용기창문부품이며 그 일부분을 현시할수 있다. 그것은 필요할 때 포함된 창문부품의 어떤 부분을 볼수 있게 마음대로 선택하게 하는 흘림띠들을 제공한다.

머리부파일

#include <qscrollview.h>

기초클라스

QFrame QObject QPaintDevice QWidget Qt

파생클라스

KApplicationTree KFileDetailView KFileIconView KHTMLView KIconCanvas KIconView KListBox KListView KSplitList KTextBrowser

QCanvasView QIconView QListBox QListView QTextBrowser QTextView

구성자

QScrollView(QWidget *parent = 0, const char *name = 0, WFlags f = 0);

메쏘드

```
virtual void addChild(QWidget *child, int x = 0, int y = 0);
bool childIsVisible(QWidget *child);
int childX(QWidget *child);
int childY(QWidget *child);
QWidget *clipper() const;
int contentsHeight() const;
void contentsToViewport(int x, int y, int &vx, int &vy);
QPoint contentsToViewport(const QPoint &);
int contentsWidth() const;
int contentsX() const;
int contentsY() const;
QWidget *cornerWidget() const;
bool dragAutoScroll() const;
void enableClipper(bool y);
ScrollBarMode hScrollBarMode() const;
QScrollBar *horizontalScrollBar() const;
QSize minimumSizeHint() const;
virtual void moveChild(QWidget *child, int x, int y);
void removeChild(QWidget *child);
void removeChild(QObject *child);
void repaintContents(int x, int y, int w, int h, bool erase = TRUE);
void repaintContents(const QRect &r, bool erase = TRUE);
void resize(int w, int h);
void resize(const QSize &);
ResizePolicy resizePolicy() const;
virtual void setCornerWidget(QWidget *);
void setDragAutoScroll(bool b);
virtual void setHScrollBarMode(ScrollBarMode);
virtual void setResizePolicy(ResizePolicy);
virtual void setVScrollBarMode(ScrollBarMode);
void show();
void showChild(QWidget *child, bool yes = TRUE);
QSize sizeHint() const;
QSizePolicy sizePolicy() const;
void updateContents(int x, int y, int w, int h);
```

```
void updateContents(const QRect &r);
ScrollBarMode vScrollBarMode() const;
QScrollBar *verticalScrollBar() const;
QWidget *viewport() const;
QSize viewportSize(int, int) const;
void viewportToContents(int vx, int vy, int &x, int &y);
QPoint viewportToContents(const QPoint &);
int visibleHeight() const;
int visibleWidth() const;
```

처리부

```
void center(int x, int y);
void center(int x, int y, float xmargin, float ymargin);
void ensureVisible(int x, int y);
void ensureVisible(int x, int y, int xmargin, int ymargin);
virtual void resizeContents(int w, int h);
void scrollBy(int dx, int dy);
virtual void setContentsPos(int x, int y);
void setEnabled(bool enable);
void updateScrollBars();
```

신호

```
void contentsMoving(int x, int y);
```

렬거형

```
enum ResizePolicy { Default, Manual, AutoOne };
enum ScrollBarMode { Auto, AlwaysOff, AlwaysOn };
```

다음 실례는 QScrollView창문부품에 QLCDNumber창문부품을 삽입한다. 그림 18-16에서 QLCDNumber창문부품의 일부를 볼수 있게 조종하는데 사용할수 있는 흘림띠들을 가지고있는 창문부품의 한 부분을 보여준다.

```
/* showscrollview.cpp */
#include <qapplication.h>
#include <qscrollview.h>
#include <qlcdnumber.h>
int main(int argc,char **argv)
{
    QApplication app(argc,argv);
    QScrollView *scrollview = new QScrollView();
    QLCDNumber *number = new QLCDNumber();
```

```
    number->setNumDigits(8);
    number->display(982.89021);
    number->setMinimumSize(600,400);
    scrollview->addChild(number);
    scrollview->show();
    app.setMainWidget(scrollview);
    return(app.exec());
}
```
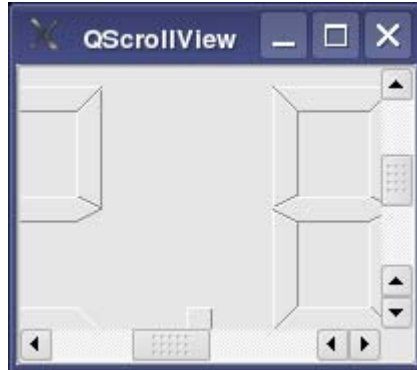
그림 18-16. 자식창문부품의 일부분을 현시하는 QScrollView창문부품

QSemiModal

QSemiModal창문부품은 같은 응용프로그람의 임의의 다른 창문에서 마우스호출을 금지함으로써 QWidget에 상반성을 추가한다.

머리부파일

#include <qsemimodal.h>

기초클라스

QObject QPaintDevice QWidget Qt

파생클라스

QProgressDialog

구성자

QSemiModal(QWidget *parent = 0, const char *name = 0, bool modal = FALSE, WFlags f = 0);

메쏘드

```
void move(int x, int y);
void move(const QPoint &p);
void resize(int w, int h);
void resize(const QSize &);
virtual void setGeometry(int x, int y, int w, int h);
```

```
virtual void setGeometry(const QRect &);
void show();
```

QSemiModal창문부품은 단지 하나의 보충적인 특성을 가지는 QWidget이다. 즉 구성자의 3번째 인수가 TRUE이면 이 응용프로그람의 다른 창문은 마우스나 건반에 응답하지 않는다. 다음 실례는 QSemiModal이 어떤 다른 창문부품으로 사용될수 있다는것을 보여주며 그림 18-17과 같이 다른 창문부품들을 포함할수 있다. 또한 이 실례의 Exit단추와 같이 모달창문부품이 사용자에게 탈출수단을 제공하는것은 아주 중요하다.

```
/* showsemimodal.cpp */
#include <qapplication.h>
#include <qpushbutton.h>
#include <qsemimodal.h>
int main(int argc,char **argv)
{
    QApplication app(argc,argv);
    QSemiModal *semimodal = new QSemiModal(0, "semimodal",  TRUE);
    QPushButton *button = new QPushButton("Exit",  semimodal);
    QObject::connect(button, SIGNAL(clicked()),&app,SLOT(quit()));
    semimodal->show();
    app.setMainWidget(semimodal);
    return(app.exec());
}
```
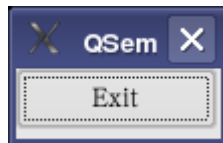
그림 18-17. QPushButton을 가진 QSemiModal창문부품

QSizeGrip

QSizeGrip창문부품은 창문의 오른쪽아래구석에 설계된 크기조절손잡이이다.

머리부파일

```
#include <qsizegrip.h>
```

기초클라스

QObject QPaintDevice QWidget Qt

구성자

```
QSizeGrip(QWidget *parent, const char *name = 0);
```

메쏘드

```
QSize  sizeHint()  const;
QSizePolicy sizePolicy() const;
```

다음 실례는 QSizeHandle를 창문부품에 추가하고 마우스로 그것을 끌어서 포함된 창문부품의 크기를 변경하는것을 보여준다. 그림 18-18은 오른쪽아래구석에 QResizeGrip창문부품의 표준위치를 보여준다.

```
/* showsizegrip.cpp */
#include <qapplication.h>
#include <qsizegrip.h>
int main(int argc,char **argv)
{
    QApplication app(argc,argv);
    QWidget *widget = new QWidget();
    widget->setMinimumSize(200,150);
    QSizeGrip *sizegrip = new QSizeGrip(widget);
    sizegrip->setGeometry(170,120,30,30);
    widget->show();
    app.setMainWidget(widget);
    return(app.exec());
}
```
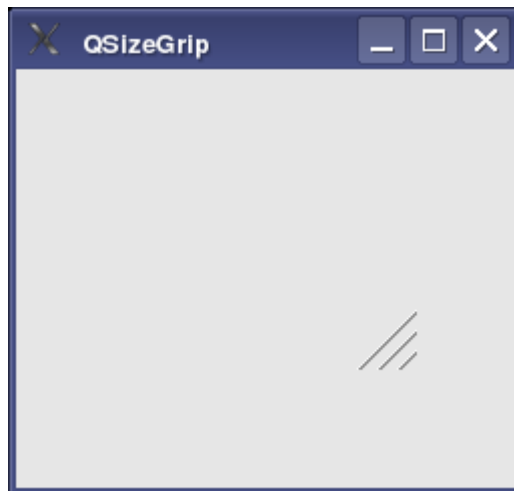
그림 18-18. 창문의 오른쪽아래구석에 있는 QSIzeGrip

QSlider

QSlider창문부품은 마우스로 2개 극값사이에서 값을 조절하는데 쓰이는 궤도와 이동가능한 활차(thumb)를 현시한다.

머리부파일

```
#include <qslider.h>
```

기초클라스

QObject QPaintDevice QRangeControl QWidget Qt

구성자

```
QSlider(QWidget *parent, const char *name = 0);
QSlider(Orientation, QWidget *parent, const char *name = 0);
QSlider(int minValue, int maxValue, int pageStep, int value, Orientation, QWidget *parent,
        const char *name = 0);
```

메쏘드

```
int lineStep() const;
int maxValue() const;
int minValue() const;
QSize minimumSizeHint() const;
Orientation orientation() const;
int pageStep() const;
void setLineStep(int);
void setMaxValue(int);
void setMinValue(int);
virtual void setOrientation(Orientation);
void setPageStep(int);
virtual void setPalette(const QPalette &);
virtual void setTickInterval(int);
virtual void setTickmarks(TickSetting);
virtual void setTracking(bool enable);
QSize sizeHint() const;
QSizePolicy sizePolicy() const;
QRect sliderRect() const;
int tickInterval() const;
TickSetting tickmarks() const;
bool tracking() const;
int value() const;
```

처리부

```
void addStep();
virtual void setValue(int);
void subtractStep();
```

신호

```
void sliderMoved(int value);
void sliderPressed();
void sliderReleased();
void valueChanged(int value);
```

렬거형

```
enum TickSetting { NoMarks=0, Above=1, Left=Above, Below=2, Right=Below, Both=3 };
```

다음 실례는 그림 18-19와 같이 수직 및 수평미끄럼띠 각각 한개씩, 그리고 눈금표식자를 현시한다.

```
/* showslider.cpp */
#include <qapplication.h>
#include <qslider.h>
int main(int argc,char **argv)
{
    QApplication app(argc,argv);
    QWidget *widget = new QWidget();
    QSlider *vslider = new QSlider(Qt::Vertical,widget);
    vslider->setTickmarks(QSlider::Left);
    vslider->setGeometry(200,0,30,200);
    QSlider *hslider = new QSlider(Qt::Horizontal,widget);
    hslider->setTickmarks(QSlider::Above);
    hslider->setGeometry(0,200,200,30);
    widget->setFixedSize(230,230);
    widget->show();
    app.setMainWidget(widget);
    return(app.exec());
}
```
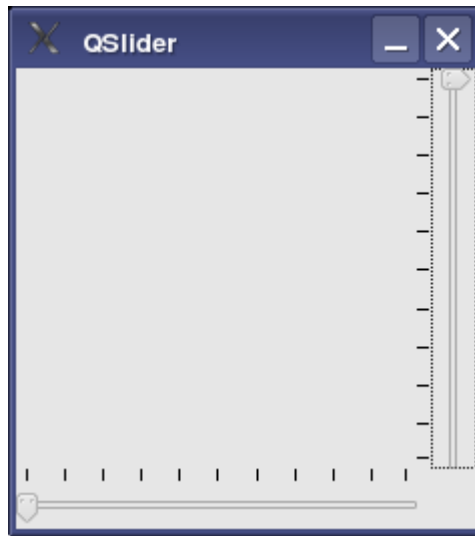
그림 18-19. 수평QSLider와 수직QSLider

QSpinBox

QSpinBox창문부품은 현재값 또는 설정값을 가지는 본문창문을 현시한다. 그리고 어떤 값이나 설정값을 다음 값으로 사용자가 절환할수 있는 2개의 단추를 가진다.

머리부파일

```
#include <qspinbox.h>
```

기초클라스

QFrame QObject QPaintDevice QRangeControl QWidget Qt

파생클라스

KIntSpinBox

구성자

```
QSpinBox(QWidget *parent = 0, const char *name = 0);
QSpinBox(int minValue, int maxValue, int step = 1, QWidget *parent = 0, const char *name = 0);
```

메쏘드

```
ButtonSymbols buttonSymbols() const;
virtual QString cleanText() const;
int lineStep() const;
int maxValue() const;
int minValue() const;
virtual QString prefix() const;
void setButtonSymbols(ButtonSymbols);
void setLineStep(int);
```

```
void setMaxValue(int);
void setMinValue(int);
virtual void setSpecialValueText(const QString &text);
virtual void setValidator(const QValidator *v);
virtual void setWrapping(bool on);
QSize sizeHint() const;
QSizePolicy sizePolicy() const;
QString specialValueText() const;
virtual QString suffix() const;
QString text() const;
const QValidator *validator() const;
int value() const;
bool wrapping() const;
```

처리부

```
virtual void setEnabled(bool);
virtual void setPrefix(const QString &text);
virtual void setSuffix(const QString &text);
virtual void setValue(int value);
virtual void stepDown();
virtual void stepUp();
```

신호

```
void valueChanged(int value);
void valueChanged(const QString &valueText);
```

렬거형

```
enum ButtonSymbols { UpDownArrows, PlusMinus };
```

다음 실례는 그림 18-20과 같이 스핀칸을 만든다. 수값은 10～100범위에서 변화할수 있고 화살단추를 선택하면 수값이 10씩 증가 또는 감소된다. 앞붙이와 뒤붙이있는 현시값을 사용자정의할수 있으며 또는 현시문자렬을 setSpecialValueText()호출로 완전히 교체할수 있다.

```
/* showspinbox.cpp */
#include <qapplication.h>
#include <qspinbox.h>
int main(int argc,char **argv)
{
    QApplication app(argc,argv);
    QSpinBox *spinbox = new QSpinBox(10,100,5);
```

```
    spinbox->show();
    app.setMainWidget(spinbox);
    return(app.exec());
}
```
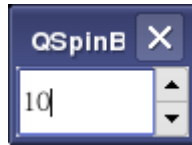
그림 18-20. 옹근수값을 선택하는데 사용된 QSpinBox

QSplitter

QSplitter창문부품은 요구에 따라 흘림띠를 가진 2개의 창문을 포함하며 마우스로 각자의 상대크기를 조절할수 있다.

머리부파일

```
#include <qsplitter.h>
```

기초클라스

QFrame QObject QPaintDevice QWidget Qt

파생클라스

KCombiView KFilePreview

구성자

```
QSplitter(QWidget *parent = 0, const char *name = 0);
QSplitter(Orientation, QWidget *parent = 0, const char *name = 0);
```

메쏘드

```
QSize minimumSizeHint() const;
void moveToFirst(QWidget *);
void moveToLast(QWidget *);
bool opaqueResize() const;
Orientation orientation() const;
void refresh();
virtual void setOpaqueResize(bool = TRUE);
virtual void setOrientation(Orientation);
virtual void setResizeMode(QWidget *w, ResizeMode);
void setSizes(QValueList < int >);
QSize sizeHint() const;
QSizePolicy sizePolicy() const;
QValueList<int> sizes() const;
```

렬거형

```
enum ResizeMode { Stretch, KeepSize, FollowSizeHint };
```

7장과 8장에 QSplitter의 실례가 있다

QStatusBar

QStatusBar창문부품은 동적으로 설정 혹은 삭제할수 있는 본문행을 현시한다. 또한 자체와 부모창문의 크기를 조절하는데 사용되는 QSizeGrip창문부품을 포함한다.

머리부파일

```
#include <qstatusbar.h>
```

기초클라스

QObject QPaintDevice QWidget Qt

파생클라스

KStatusBar

구성자

```
QStatusBar(QWidget *parent = 0, const char *name = 0);
```

메쏘드

```
void addWidget(QWidget *, int stretch = 0, bool = FALSE);
bool isSizeGripEnabled() const;
void removeWidget(QWidget *);
void setSizeGripEnabled(bool);
```

처리부

```
void clear();
void message(const QString &);
void message(const QString &, int);
```

다음 실례는 그림 18-21과 같이 QStatusBar창문부품을 가진 창문을 현시한다. 이 종류의 창문부품은 보통 창문의 바닥에 포함된다.

```
/* showstatusbar.cpp */
#include <qapplication.h>
#include <qstatusbar.h>
int main(int argc,char **argv)
{
    QApplication app(argc,argv);
    QStatusBar *statusbar = new QStatusBar();
    statusbar->setSizeGripEnabled(TRUE);
    statusbar->message("The QSatusBar widget");
```

```
    statusbar->show();
    app.setMainWidget(statusbar);
    return(app.exec());
}
```
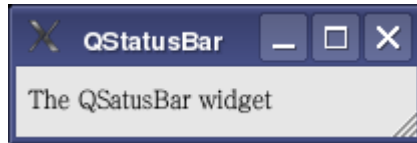
그림 18-21. QSizeGrip창문부품이 허용되는 QStatusBar창문부품

QTabBar

QTabBar창문부품은 마우스로 개별적으로 선택할수 있는 타브행을 현시한다.

머리부파일

```
#include <qtabbar.h>
```

기초클라스

QObject QPaintDevice QWidget Qt

구성자

```
QTabBar(QWidget *parent = 0, const char *name = 0);
```

메쏘드

```
virtual int addTab(QTab *);
int currentTab() const;
virtual int insertTab(QTab *, int index = - 1);
bool isTabEnabled(int) const;
int keyboardFocusTab() const;
virtual void layoutTabs();
virtual void removeTab(QTab *);
virtual void setShape(Shape);
virtual void setTabEnabled(int, bool);
Shape shape() const;
void show();
QSize sizeHint() const;
QSizePolicy sizePolicy() const;
QTab *tab(int);
```

처리부

```
virtual void setCurrentTab(int);
virtual void setCurrentTab(QTab *);
```

신호

void selected(int);

렬거형

enum Shape { RoundedAbove, RoundedBelow, TriangularAbove, TriangularBelow };

다음 실례는 그림 18-22와 같이 4개 타브와 QTabBar를 만든다. 띠의 오른쪽끝에 부모창문이 타브들을 모두 포함하지 못할 때 나타나는 스핀단추가 있다. addTab()호출은 타브를 선택할 때마다 selected()신호를 발생하는데 사용되는 옹근수식별번호를 돌려준다.

```
/* showtabbar.cpp */
#include <qapplication.h>
#include <qtabbar.h>
int main(int argc,char **argv)
{
    QApplication app(argc,argv);
    QTabBar *tabbar = new QTabBar();
    tabbar->addTab(new QTab("First"));
    tabbar->addTab(new QTab("Second"));
    tabbar->addTab(new QTab("Third"));
    tabbar->addTab(new QTab("Fourth"));
    tabbar->show();
    app.setMainWidget(tabbar);
    return(app.exec());
}
```
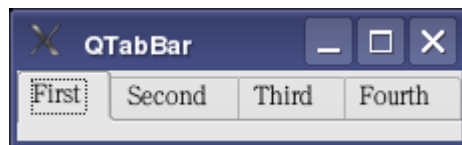
그림 18-22. 4개의 타브와 하나의 수평스핀단추를 가진 QTabBar

QTabDialog

QTabDialog창문부품은 포함된 창문부품들을 탄창에 보관하는 용기이다.

머리부파일

#include <qtabdialog.h>

기초클라스

QDialog QObject QPaintDevice QWidget Qt

구성자

QTabDialog(QWidget *parent = 0, const char *name = 0, bool modal = FALSE, WFlags f = 0);

메쏘드

```
void addTab(QWidget *, const QString &);
void addTab(QWidget *child, const QIconSet &iconset,
const QString &label);
void addTab(QWidget *, QTab *);
void changeTab(QWidget *, const QString &);
void changeTab(QWidget *child, const QIconSet &iconset, const QString &label);
QWidget *currentPage() const;
bool hasApplyButton() const;
bool hasCancelButton() const;
bool hasDefaultButton() const;
bool hasHelpButton() const;
bool hasOkButton() const;
void insertTab(QWidget *, const QString &, int index = - 1);
void insertTab(QWidget *child, const QIconSet &iconset, const QString &label, int index = - 1);
void insertTab(QWidget *, QTab *, int index = - 1);
bool isTabEnabled(QWidget *) const;
bool isTabEnabled(const char *) const;
void removePage(QWidget *);
void setApplyButton(const QString &text);
void setApplyButton();
void setCancelButton(const QString &text);
void setCancelButton();
void setDefaultButton(const QString &text);
void setDefaultButton();
void setFont(const QFont &font);
void setHelpButton(const QString &text);
void setHelpButton();
void setOKButton(const QString &text = QString::null);
void setOkButton(const QString &text);
void setOkButton();
void setTabEnabled(QWidget *, bool);
void setTabEnabled(const char *, bool);
void show();
void showPage(QWidget *);
QString tabLabel(QWidget *);
```

신호

void aboutToShow();

void applyButtonPressed();

void cancelButtonPressed();

void defaultButtonPressed();

void helpButtonPressed();

void selected(const QString &);

QTabDialog창문부품의 실례는 5장에 있다.

QTabWidget

QTabWidget는 마우스로 개별적으로 선택할수 있는 타브들의 렬을 현시한다.

머리부파일

#include <qtabwidget.h>

기초클라스

QObject QPaintDevice QWidget Qt

구성자

QTabWidget(QWidget *parent, const char *name, WFlags f);

QTabWidget(QWidget *parent = 0, const char *name = 0);

메쏘드

void addTab(QWidget *, const QString &);

void addTab(QWidget *child, const QIconSet &iconset, const QString &label);

void addTab(QWidget *, QTab *);

void changeTab(QWidget *, const QString &);

void changeTab(QWidget *child, const QIconSet &iconset, const QString &label);

QWidget *currentPage() const;

void insertTab(QWidget *, const QString &, int index = - 1);

void insertTab(QWidget *child, const QIconSet &iconset, const QString &label, int index = - 1);

void insertTab(QWidget *, QTab *, int index = - 1);

bool isTabEnabled(QWidget *) const;

int margin() const;

QSize minimumSizeHint() const;

void removePage(QWidget *);

void setMargin(int);

void setTabEnabled(QWidget *, bool);

void setTabPosition(TabPosition);

```
void showPage(QWidget *);
QSize sizeHint() const;
QString tabLabel(QWidget *);
TabPosition tabPosition() const;
```

신호

```
void selected(const QString &);
```

렬거형

```
enum TabPosition { Top, Bottom };
```

다음 실례는 그림 18-23과 같이 4개 창문부품을 가지는 QTabWidget용기를 만든다. 띠의 오른쪽끝에 부모창문이 타브들을 모두 포함하지 못할 때 나타나는 스핀단추가 있다.

```
/* showtabwidget.cpp */
#include <qapplication.h>
#include <qtabwidget.h>
#include <qlabel.h>
int main(int argc,char **argv)
{
    QLabel *label;
    QApplication app(argc,argv);
    QTabWidget *tabwidget = new QTabWidget();
    label = new QLabel("The First Widget Label",  tabwidget);
    tabwidget->addTab(label, "First");
    label = new QLabel("The Second Widget Label",  tabwidget);
    tabwidget->addTab(label, "Second");
    label = new QLabel("The Third Widget Label",  tabwidget);
    tabwidget->addTab(label, "Third");
    label = new QLabel("The Fourth Widget Label",  tabwidget);
    tabwidget->addTab(label, "Fourth");
    tabwidget->show();
    app.setMainWidget(tabwidget);
    return(app.exec());
}
```

그림 18-23. 4개 타브와 하나의 수평스핀단추를 가지는 QtabWidget

QTextBrowser

QTextBrowser창문부품은 창문에 본문을 현시하고 일련의 위치결정메쏘드들을 제공한다. 기본본문은 QTextView로부터 계승된다. 선택된 부분들을 얻을수 있다.

머리부파일

```
#include <qtextbrowser.h>
```

기초클라스

QFrame QObject QPaintDevice QScrollView QTextView QWidget Qt

파생클라스

KTextBrowser

구성자

```
QTextBrowser(QWidget *parent = 0, const char *name = 0);
```

메쏘드

```
void scrollToAnchor(const QString &name);
virtual void setSource(const QString &name);
void setText(const QString &contents,
const QString &context = QString::null);
QString source() const;
```

처리부

```
virtual void backward();
virtual void forward();
virtual void home();
```

신호

```
void backwardAvailable(bool);
void forwardAvailable(bool);
void highlighted(const QString &);
```

```
        void textChanged();
```

다음 실례는 그림 18-24와 같이 본문행들을 현시한다.

```
/* showtextbrowser.cpp */
#include <qapplication.h>
#include <qtextbrowser.h>
char text[] =
    "This is the text being displayed\n"
    "by the text browser. Both vertical\n"
    "and horizontal scroll bars will\n"
    "appear as necessary.";
int main(int argc,char **argv)
{
    QApplication app(argc,argv);
    QTextBrowser *textbrowser = new QTextBrowser();
    textbrowser->show();
    textbrowser->setText(QString(text));
    app.setMainWidget(textbrowser);
    return(app.exec());
}
```
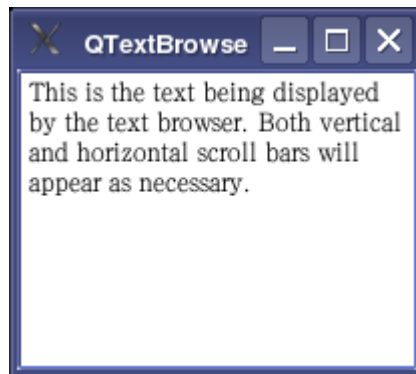
그림 18-24. 본문을 현시하는 QTextView

QTextView

QTextView창문부품은 창문에 본문을 현시한다.

머리부파일

```
#include <qtextview.h>
```

기초클라스

QFrame QObject QPaintDevice QScrollView QWidget Qt

파생클라스

KTextBrowser QTextBrowser

구성자

```
QTextView(QWidget *parent = 0, const char *name = 0);
QTextView(const QString &text, const QString &context = QString::null, QWidget *parent = 0,
    const char *name = 0);
```

메쏘드

```
void append(const QString &text);
virtual QString context() const;
QString documentTitle() const;
bool hasSelectedText() const;
int heightForWidth(int w) const;
const QColor & linkColor() const;
bool linkUnderline() const;
QMimeSourceFactory *mimeSourceFactory() const;
const QBrush & paper();
const QBrush & paper() const;
const QColorGroup & paperColorGroup() const;
QString selectedText() const;
void setLinkColor(const QColor &);
void setLinkUnderline(bool);
void setMimeSourceFactory(QMimeSourceFactory *factory);
void setPaper(const QBrush &pap);
void setPaperColorGroup(const QColorGroup &colgrp);
void setStyleSheet(QStyleSheet *styleSheet);
virtual void setText(const QString &text, const QString &context);
void setText(const QString &text);
void setTextFormat(TextFormat);
QStyleSheet *styleSheet() const;
virtual QString text() const;
TextFormat textFormat() const;
```

처리부

```
void copy();
void selectAll();
```

다음 실례는 그림 18-25과 같이 본문블로크를 현시한다.

```
/* showtextview.cpp */
```

```
#include <qapplication.h>
#include <qtextview.h>
char text[] =
    "This is the text being displayed\n"
    "by the text view. Both vertical\n"
    "and horizontal scroll bars will\n"
    "appear as necessary.";
int main(int argc,char **argv)
{
    QApplication app(argc,argv);
    QTextView *textview = new QTextView(text);
    textview->show();
    app.setMainWidget(textview);
    return(app.exec());
}
```
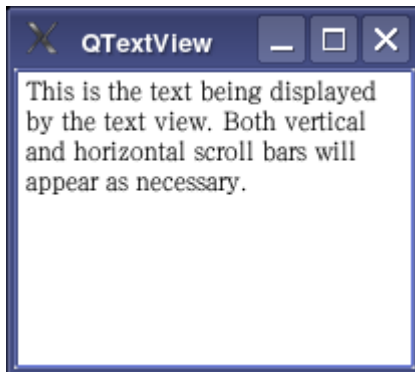
그림 18-25. 본문을 현시하는 QTextView창문부품

QToolBar

QToolBar창문부품은 도구를 포함하는 도구띠이다. 도구띠조종은 창문부품일수 있으나 대체로 본문대신에 그림기호를 가지는 작은 단추이다.

머리부파일

```
#include <qtoolbar.h>
```

기초클라스

QObject QPaintDevice QWidget Qt

구성자

```
QToolBar(const QString &label, QMainWindow *,
        QMainWindow::ToolBarDock = QMainWindow::Top,
```

```
          bool newLine = FALSE, const char *name = 0);
      QToolBar(const QString &label, QMainWindow *, QWidget *, bool newLine = FALSE,
          const char *name = 0, WFlags f = 0);
      QToolBar(QMainWindow *parent = 0, const char *name = 0);
```

메쏘드

```
      void addSeparator();
      void clear();
      bool event(QEvent *e);
      bool eventFilter(QObject *, QEvent *);
      void hide();
      bool isHorizontalStretchable() const;
      bool isVerticalStretchable() const;
      QString label() const;
      QMainWindow *mainWindow();
      QSize minimumSize() const;
      QSize minimumSizeHint() const;
      Orientation orientation() const;
      void setHorizontalStretchable(bool b);
      virtual void setLabel(const QString &);
      virtual void setOrientation(Orientation);
      virtual void setStretchableWidget(QWidget *);
      void setVerticalStretchable(bool b);
      void show();
```

신호

```
      void orientationChanged(Orientation);
```

다음 실례는 QToolBar를 창조하여 QMainWindow창문부품에 련결한다. 그림 18-26은 꼭대기에 도구띠를 보여주지만 QMainWindow창문부품은 네 변의 어디에나 도구띠를 이동할수 있게 한다. 도구띠에는 하나의 표준QPushButton창문부품과 3개의 QToolButton창문부품이 포함된다. QToolButton창문부품들중 하나는 화살표를 현시하고 다른 두개는 픽스매프들을 현시한다. 이것은 실례를 간단하게 하지만 단추들에는 보통 단추가 능동으로 될 때 호출되는 처리부메쏘드들이 할당된다.

```
/* showtoolbar.cpp */
#include <qapplication.h>
#include <qmainwindow.h>
#include <qtoolbar.h>
#include <qtoolbutton.h>
```

```
#include <qpushbutton.h>
int main(int argc,char **argv)
{
    QApplication app(argc,argv);
    QMainWindow *mainwindow = new QMainWindow();
    QToolBar *toolbar = new QToolBar("Bar",  mainwindow);
    new QPushButton("Button",  toolbar);
    QPixmap idea("idea.png");
    new QToolButton(idea, "Idea","Group",0,0,toolbar);
    QPixmap flag("flag.png");
    new QToolButton(flag, "Flag","Group",0,0,toolbar);
    new QToolButton(Qt::UpArrow,toolbar);
    mainwindow->show();
    app.setMainWidget(mainwindow);
    return(app.exec());
}
```
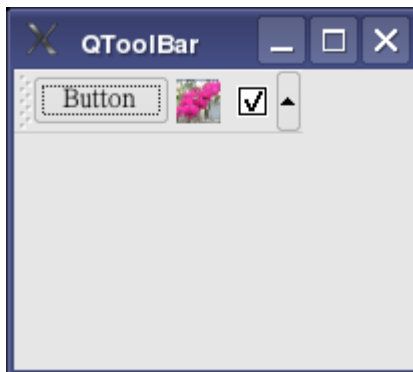
그림 18-26. 4개 단추를 가지는 QToolBar

QToolButton

QToolButton창문부품은 도구띠의 성원으로 포함되도록 설계된 특수한 단추창문부품이다. 그것은 본문, 픽스매프 또는 본문과 픽스매프를 동시에 현시할수 있다.

머리부파일

```
#include <qtoolbutton.h>
```

기초클라스

QButton QObject QPaintDevice QWidget Qt

구성자

```
QToolButton(QWidget *parent, const char *name = 0);
```

```
QToolButton(const QPixmap &pm, const QString &textLabel, const QString &grouptext,
        QObject *receiver, const char *slot, QToolBar *parent, const char *name = 0);
QToolButton(const QIconSet &s, const QString &textLabel, const QString &grouptext,
        QObject *receiver, const char *slot, QToolBar *parent, const char *name = 0);
QToolButton(ArrowType type, QWidget *parent, const char *name = 0);
```

메쏘드

```
bool autoRaise() const;
QIconSet iconSet(bool on = FALSE) const;
QIconSet offIconSet() const;
QIconSet onIconSet() const;
QPopupMenu *popup() const;
int popupDelay() const;
void setAutoRaise(bool enable);
virtual void setIconSet(const QIconSet &, bool on = FALSE);
void setOffIconSet(const QIconSet &);
void setOnIconSet(const QIconSet &);
void setPopup(QPopupMenu *popup);
void setPopupDelay(int delay);
QSize sizeHint() const;
QSizePolicy sizePolicy() const;
QString textLabel() const;
bool usesBigPixmap() const;
bool usesTextLabel() const;
```

처리부

```
virtual void setOn(bool enable);
virtual void setTextLabel(const QString &, bool);
void setTextLabel(const QString &);
virtual void setToggleButton(bool enable);
virtual void setUsesBigPixmap(bool enable);
virtual void setUsesTextLabel(bool enable);
void toggle();
```

QToolButton의 실례는 이 장의 QToolBar에 있다.

QVBox

QVBox는 자식창문부품들을 겹쌓여 배치하는 간단한 용기이다.

머리부파일

#include <qvbox.h>

기초클라스

QFrame QHBox QObject QPaintDevice QWidget Qt

파생클라스

KCharSelect

구성자

QVBox(QWidget *parent = 0, const char *name = 0, WFlags f = 0, bool allowLines = TRUE);

다음 실례는 제일웃준위창문부품으로서 QVBox를 사용한다. 그것은 자식창문부품으로서 4개의 QLabel창문부품을 가지고있다. 그림 18-27과 같이 매개 표식자는 5화소너비의 공간을 두고 현시된다.

```
/* showvbox.cpp */
#include <qapplication.h>
#include <qvbox.h>
#include <qlabel.h>
int main(int argc,char **argv)
{
    QApplication app(argc,argv);
    QVBox *vbox = new QVBox();
    new QLabel("First",  vbox);
    new QLabel("Second",  vbox);
    new QLabel("Third",  vbox);
    new QLabel("Fourth",  vbox);
    vbox->setSpacing(5);
    vbox->show();
    app.setMainWidget(vbox);
    return(app.exec());
}
```
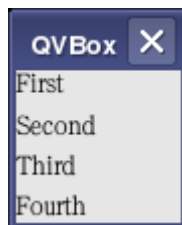
그림 18-27. QVBox에 의해 현시된 표식자들

QVButtonGroup

QVButtonGroup는 수직렬에 단추모임을 배치하는 용기창문부품이다.

머리부파일

```
#include <qvbuttongroup.h>
```

기초클라스

QButtonGroup QFrame QGroupBox QObject QPaintDevice QWidget Qt

구성자

```
QVButtonGroup(QWidget *parent = 0, const char *name = 0);
QVButtonGroup(const QString &title, QWidget *parent = 0, const char *name = 0);
```

7장에 QVButtonGroup의 실례가 있다

QVGroupBox

QVGroupBox는 수직렬에 창문부품모임을 배치하는 용기창문부품이다.

머리부파일

```
#include <qvgroupbox.h>
```

기초클라스

QFrame QGroupBox QObject QPaintDevice QWidget Qt

구성자

```
QVGroupBox(QWidget *parent = 0, const char *name = 0);
QVGroupBox(const QString &title, QWidget *parent = 0, const char *name = 0);
```

다음 실례는 QVGroupBox창문부품안에 4개 표식자를 포함한다. 그림 18-28과 같이 QVGroupBox창문부품은 QFrame을 계승하므로 포함된 창문부품들의 둘레에 경계선을 현시하고 마음대로 제목을 현시할수 있다.

```
/* showvgroupbox.cpp */
#include <qapplication.h>
#include <qvgroupbox.h>
#include <qlabel.h>
int main(int argc,char **argv)
{
    QApplication app(argc,argv);
    QVGroupBox *vgroupbox = new QVGroupBox();
    new QLabel("First",  vgroupbox);
    new QLabel("Second",  vgroupbox);
    new QLabel("Third",  vgroupbox);
```

```
    new QLabel("Fourth",  vgroupbox);
    vgroupbox->setTitle("Group Box Title");
    vgroupbox->show();
    app.setMainWidget(vgroupbox);
    return(app.exec());
}
```
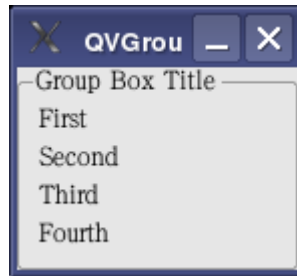
그림 18-28. QVGroupBox에 의해 포함된 4개 단추

QWidget

QWidget클라스는 모든 사용자대면부클라스의 기초클라스이다.

머리부파일

```
#include <qwidget.h>
```

기초클라스

QObject QPaintDevice Qt

파생클라스

QWidget를 기초클라스로 사용하는 Qt와 KDE의 매개 창문부품.

구성자

```
QWidget(QWidget *parent = 0, const char *name = 0, WFlags f = 0);
```

메쏘드

```
bool acceptDrops() const;
virtual void adjustSize();
bool autoMask() const;
const QColor & backgroundColor() const;
BackgroundMode backgroundMode() const;
BackgroundOrigin backgroundOrigin() const;
const QPixmap *backgroundPixmap() const;
QSize baseSize() const;
QString caption() const;
QRect childrenRect() const;
```

```
QRegion childrenRegion() const;
void clearMask();
virtual bool close(bool alsoDelete);
const QColorGroup & colorGroup() const;
const QCursor & cursor() const;
virtual bool customWhatsThis() const;
void drawText(int x, int y, const QString &);
void drawText(const QPoint &, const QString &);
void erase();
void erase(int x, int y, int w, int h);
void erase(const QRect &);
void erase(const QRegion &);
static QWidget *find(WId);
FocusPolicy focusPolicy() const;
QWidget *focusProxy() const;
QWidget *focusWidget() const;
QFont font() const;
QFontInfo fontInfo() const;
QFontMetrics fontMetrics() const;
PropagationMode fontPropagation() const;
const QColor & foregroundColor() const;
QRect frameGeometry() const;
QSize frameSize() const;
const QRect & geometry() const;
void grabKeyboard();
void grabMouse();
void grabMouse(const QCursor &);
bool hasFocus() const;
bool hasMouseTracking() const;
int height() const;
virtual int heightForWidth(int) const;
const QPixmap *icon() const;
QString iconText() const;
bool isActiveWindow() const;
bool isDesktop() const;
bool isEnabled() const;
```

```
bool isEnabledTo(QWidget *) const;
bool isEnabledToTLW() const;
bool isFocusEnabled() const;
bool isMinimized() const;
bool isModal() const;
bool isPopup() const;
bool isTopLevel() const;
bool isUpdatesEnabled() const;
bool isVisible() const;
bool isVisibleTo(QWidget *) const;
bool isVisibleToTLW() const;
static QWidget *keyboardGrabber();
QLayout *layout() const;
QPoint mapFromGlobal(const QPoint &) const;
QPoint mapFromParent(const QPoint &) const;
QPoint mapToGlobal(const QPoint &) const;
QPoint mapToParent(const QPoint &) const;
int maximumHeight() const;
QSize maximumSize() const;
int maximumWidth() const;
QRect microFocusHint() const;
int minimumHeight() const;
QSize minimumSize() const;
virtual QSize minimumSizeHint() const;
int minimumWidth() const;
static QWidget *mouseGrabber();
const QPalette & palette() const;
PropagationMode palettePropagation() const;
QWidget *parentWidget() const;
QPoint pos() const;
void recreate(QWidget *parent, WFlags f, const QPoint &p, bool showIt = FALSE);
QRect rect() const;
void releaseKeyboard();
void releaseMouse();
virtual void reparent(QWidget *parent, WFlags, const QPoint &, bool showIt = FALSE);
void reparent(QWidget *parent, const QPoint &, bool showIt = FALSE);
```

```
void scroll(int dx, int dy);
void scroll(int dx, int dy, const QRect &);
virtual void setAcceptDrops(bool on);
virtual void setActiveWindow();
virtual void setAutoMask(bool);
virtual void setBackgroundColor(const QColor &);
virtual void setBackgroundMode(BackgroundMode);
void setBackgroundOrigin(BackgroundOrigin);
virtual void setBackgroundPixmap(const QPixmap &);
void setBaseSize(const QSize &);
void setBaseSize(int basew, int baseh);
virtual void setCursor(const QCursor &);
void setFixedHeight(int h);
void setFixedSize(const QSize &);
void setFixedSize(int w, int h);
void setFixedWidth(int w);
virtual void setFocusPolicy(FocusPolicy);
virtual void setFocusProxy(QWidget *);
virtual void setFont(const QFont &);
void setFont(const QFont &, bool iReallyMeanIt);
virtual void setFontPropagation(PropagationMode);
virtual void setMask(const QBitmap &);
virtual void setMask(const QRegion &);
void setMaximumHeight(int maxh);
void setMaximumSize(const QSize &);
virtual void setMaximumSize(int maxw, int maxh);
void setMaximumWidth(int maxw);
void setMinimumHeight(int minh);
void setMinimumSize(const QSize &);
virtual void setMinimumSize(int minw, int minh);
void setMinimumWidth(int minw);
void setName(const char *name);
virtual void setPalette(const QPalette &);
void setPalette(const QPalette &, bool iReallyMeanIt);
virtual void setPalettePropagation(PropagationMode);
void setSizeIncrement(const QSize &);
```

virtual void setSizeIncrement(int w, int h);

void setStyle(QStyle *);

static void setTabOrder(QWidget *, QWidget *);

QSize size() const;

virtual QSize sizeHint() const;

QSize sizeIncrement() const;

virtual QSizePolicy sizePolicy() const;

QStyle & style() const;

bool testWFlags(WFlags n) const;

bool testWState(uint n) const;

QWidget *topLevelWidget() const;

virtual void unsetCursor();

void unsetFont();

void unsetPalette();

void updateGeometry();

QRect visibleRect() const;

int width() const;

WId winId() const;

static QWidgetMapper *wmapper();

int x() const;

int y() const;

처리부

void clearFocus();

bool close();

void constPolish() const;

virtual void hide();

void iconify();

void lower();

virtual void move(int x, int y);

void move(const QPoint &);

virtual void polish();

void raise();

void repaint();

void repaint(bool erase);

void repaint(int x, int y, int w, int h, bool erase = TRUE);

void repaint(const QRect &, bool erase = TRUE);

void repaint(const QRegion &, bool erase = TRUE);

virtual void resize(int w, int h);

void resize(const QSize &);

virtual void setCaption(const QString &);

virtual void setEnabled(bool);

virtual void setFocus();

virtual void setGeometry(int x, int y, int w, int h);

virtual void setGeometry(const QRect &);

virtual void setIcon(const QPixmap &);

virtual void setIconText(const QString &);

virtual void setMouseTracking(bool enable);

virtual void setUpdatesEnabled(bool enable);

virtual void show();

void showFullScreen();

virtual void showMaximized();

virtual void showMinimized();

virtual void showNormal();

void update();

void update(int x, int y, int w, int h);

void update(const QRect &);

렬거형

enum BackgroundMode { FixedColor, FixedPixmap, NoBackground, PaletteForeground, PaletteButton, PaletteLight, PaletteMidlight, PaletteDark, PaletteMid, PaletteText, PaletteBrightText, PaletteBase, PaletteBackground, PaletteShadow, PaletteHighlight, PaletteHighlightedText };

enum PropagationMode { NoChildren, AllChildren, SameFont, SamePalette=SameFont };

enum FocusPolicy { NoFocus=0, TabFocus=0x1, ClickFocus=0x2, StrongFocus=0x3, WheelFocus=0x7 };

enum BackgroundOrigin { WidgetOrigin, ParentOrigin };

QWidgetStack

QWidgetStack창문부품은 한번에 1개 창문부품만 현시하는 용기이다.

머리부파일

#include <qwidgetstack.h>

기초클라스

QFrame QObject QPaintDevice QWidget Qt

구성자

QWidgetStack(QWidget *parent = 0, const char *name = 0);

메쏘드

void addWidget(QWidget *, int);

int id(QWidget *) const;

QSize minimumSizeHint() const;

void removeWidget(QWidget *);

void setFrameRect(const QRect &);

void show();

QSize sizeHint() const;

QWidget *visibleWidget() const;

QWidget *widget(int) const;

처리부

void raiseWidget(int);

void raiseWidget(QWidget *);

신호

void aboutToShow(int);

void aboutToShow(QWidget *);

다음 실례는 QWidgetStack에 3개의 QPushButton창문부품을 창조하여 추가한다. 그림 18-29와 같이 현시된 유일한 창문부품은 raiseWidget()호출에 사용된 식별번호를 가진 창문부품이다.

```
/* showwidgetstack.cpp */
#include <qapplication.h>
#include <qwidgetstack.h>
#include <qpushbutton.h>
int main(int argc,char **argv)
{
    QPushButton *button;
    QApplication app(argc,argv);
    QWidgetStack *widgetstack = new QWidgetStack();
    button = new QPushButton("First Button",  widgetstack);
    widgetstack->addWidget(button,1);
    button = new QPushButton("Second Button",  widgetstack);
    widgetstack->addWidget(button,2);
    button = new QPushButton("Third Button",  widgetstack);
    widgetstack->addWidget(button,3);
```

```
    widgetstack->raiseWidget(2);
    widgetstack->show();
    app.setMainWidget(widgetstack);
    return(app.exec());
}
```

그림 18-29. 그 창문부품들중 하나를 현시하는 QWidgetStack창문부품

QWizard

QWizard창문부품은 여러 단계의 대화칸을 창조하는데 사용한다. 매개 걸음은 하나의 창문으로 이루어져있다. QWizard창문부품은 페지화기구와 조종단추들을 제공한다.

머리부파일

```
#include <qwizard.h>
```

기초클라스

QDialog QObject QPaintDevice QWidget Qt

파생클라스

KWizard

구성자

```
QWizard(QWidget *parent = 0, const char *name = 0, bool modal = FALSE, WFlags f = 0);
```

메쏘드

```
virtual void addPage(QWidget *, const QString &);
virtual bool appropriate(QWidget *) const;
QPushButton *backButton() const;
QPushButton *cancelButton() const;
QWidget *currentPage() const;
bool eventFilter(QObject *, QEvent *);
QPushButton *finishButton() const;
QPushButton *helpButton() const;
QPushButton *nextButton() const;
QWidget *page(int pos) const;
int pageCount() const;
virtual void removePage(QWidget *);
```

```
virtual void setAppropriate(QWidget *, bool);
void setFont(const QFont &font);
void show();
virtual void showPage(QWidget *);
QString title(QWidget *) const;
```

처리부

```
virtual void setBackEnabled(QWidget *, bool);
virtual void setFinish(QWidget *, bool);
virtual void setFinishEnabled(QWidget *, bool);
virtual void setHelpEnabled(QWidget *, bool);
virtual void setNextEnabled(QWidget *, bool);
```

신호

```
void helpClicked();
```

다음 실례는 그림 18-30에 보여주는 빈 QWizard창문부품을 현시한다.

```
/* showwizard.cpp */
#include <qapplication.h>
#include <qwizard.h>
int main(int argc,char **argv)
{
    QPushButton *button;
    QApplication app(argc,argv);
    QWizard *wizard = new QWizard();
    wizard->show();
    app.setMainWidget(wizard);
    return(app.exec());
}
```

그림 18-30. QWizard창문부품

요 약

이 장은 모든 Qt창문부품들을 자모순으로 제공하였다. 매개 창문부품은 다음과 같이 목록화하였다.

· 창문부품의 실례들을 만드는데 사용할수 있는 구성자들.

· 창문부품이 정의되는 머리부파일의 이름

· 창문부품이 기능을 계승하는 모든 기초클라스들

· 창문부품이 기능을 넘겨주는 모든 파생클라스들

· 1개 창문부품의 사건을 다른 창문부품의 메쏘드호출에 련결하는데 사용되는 처리부와 신호들

· 응용프로그람에 사용할수 있는 공개메쏘드

다음 장은 KDE창문부품들을 설명한다. KDE창문부품은 QWidget로부터 계승된 KDE클라스이다.

제19장. KDE의 창문부품

학습내용

매개 창문부품의 구성자들, 필요한 머리부파일, 기초클라스와 파생클라스들, 처리부와 신호들 그리고 모든 공개메쏘드들

창문부품의 작은 실례들.

이 장은 모든 창문부품들을 자모순으로 제시한다. 창문부품은 현시가능한 창문을 포함하는 클라스이다. KDE에서 현시가능한 창문을 가지는 모든 클라스는 QWidget클라스로부터 창문기능을 계승한다. 더우기 대부분의 창문부품은 대화칸이다. 즉 QWidget를 계승하는 QDialog로부터 계승된다. 유일한 차이는 창문부품은 부모를 가져야 하고 다른 창문안에 현시되며 대화칸은 그 자체의 제일 웃준위창문을 가지고있다는것이다.

매개 창문부품은 머리부파일의 이름, 모든 기초클라스들의 이름, 모든 KDE와 Qt파생클라스들의 이름, 공개메쏘드, 처리부, 신호 그리고 렬거형들로 이루어진다.

일부 기초 및 파생클라스들을 제외한 모든 창문부품들은 실례가 있다. 매개 창문부품에 제공된 실례코드는 창문부품의 현시가능한 폼을 만든다. 일부 창문부품들은 제일 웃준위창문부품으로서 지정되고 기본창문으로 현시된다. 그러나 일부 특수창문부품은 특수한 환경에서 포함된다.

KAboutContainer

KAboutContainer는 자체의 About칸을 구성하는 끌격창문부품이다.

머리부파일

```
#include <kaboutdialog.h>
```

기초클라스

QFrame QObject QPaintDevice QWidget Qt

구성자

```
KAboutContainer(QWidget *parent = 0, const char *name = 0, int margin = 0, int spacing = 0,
    int childAlignment = AlignCenter, int innerAlignment = AlignCenter);
```

메쏘드

```
void addImage(const QString &fileName, int alignment = AlignLeft);
void addPerson(const QString &name, const QString &email, const QString &url, const QString &tas
k,
    bool showHeader = false, bool showframe = false, bool showBold = false);
void addTitle(const QString &title, int alignment = AlignLeft, bool showframe = false,
    bool showBold = false);
void addWidget(QWidget *widget);
```

```
virtual QSize minimumSizeHint(void) const;
virtual QSize sizeHint(void) const;
```

신호

```
void mailClick(const QString &name, const QString &address);
void urlClick(const QString &url);
```

KAboutContainer창문부품은 기정으로 창문에 아무것도 현시하지 않지만 자기 응용프로그람이 필요한만큼 많은 요소들을 추가하는 메쏘드들의 모임을 제공한다. 다음의 실례는 그림 19-1에 보여주는 창문을 만들고 중심에 본문과 개인정보블로크를 추가한다.

```
/* showaboutcontainer.cpp */
#include <kapplication.h>
#include <kaboutdialog.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "aboutcontainer");
    KAboutContainer *aboutcontainer = new KAboutContainer();
    aboutcontainer->addTitle("A title is a line (or block)\n"
        "of text that is stored in the window.\n",
        Qt::AlignCenter);
    aboutcontainer->addPerson("Phillip Space", "phil@nobody.com", "http://www.belugalake.com",
        "Responsible for coloring pixels");
    aboutcontainer->show();
    app.setMainWidget(aboutcontainer);
    return(app.exec());
}
```
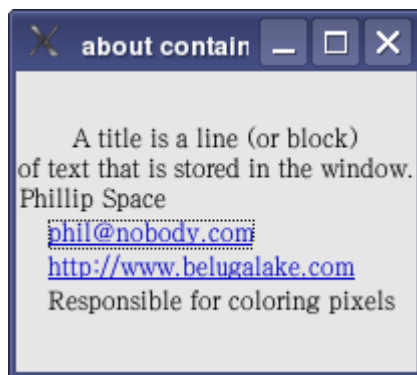
그림 19-1. 두 항목을 현시하는 KAboutContainer창문부품

KAboutContainerBase

KAboutContainerBase창문부품은 하나이상의 KAboutContainer창문부품들을 적당한 장소에 배치하고 현시할수 있다.

머리부파일

```
#include <kaboutdialog.h>
```

기초클라스

QObject QPaintDevice QWidget Qt

구성자

```
KAboutContainerBase(int layoutType, QWidget *parent = 0, char *name = 0);
```

메쏘드

```
KAboutContainer *addContainer(int childAlignment, int innerAlignment);
KAboutContainer *addContainerPage(const QString &title, int childAlignment = AlignCenter,
        int innerAlignment = AlignCenter);
QFrame *addEmptyPage(const QString &title);
QFrame *addTextPage(const QString &title, const QString &text, bool richText = false,
        int numLines = 10);
void setImage(const QString &fileName);
void setImageBackgroundColor(const QColor &color);
void setImageFrame(bool state);
void setProduct(const QString &appName, const QString &version, const QString &author,
        const QString &year);
void setTitle(const QString &title);
virtual void show(void);
virtual QSize sizeHint(void) const;
```

처리부

```
virtual void slotMailClick(const QString &name, const QString &address);
virtual void slotMouseTrack(int mode, const QMouseEvent *e);
virtual void slotUrlClick(const QString &url);
```

신호

```
void mailClick(const QString &name, const QString &address);
void mouseTrack(int mode, const QMouseEvent *e);
void urlClick(const QString &url);
```

렬거형

```
enum LayoutType { AbtPlain=0x0001, AbtTabbed=0x0002, AbtTitle=0x0004, AbtImageLeft=0x0008,
    AbtImageRight=0x0010, AbtImageOnly=0x0020, AbtProduct=0x0040,
```

AbtKDEStandard=AbtTabbed|AbtTitle|AbtImageLeft, AbtAppStandard=AbtTabbed|AbtTitle|AbtProduct,

AbtImageAndTitle=AbtPlain|AbtTitle|AbtImageOnly };

KAboutContainerBase창문부품을 사용하는 실례는 5장에 있다(KAboutDialog에서).

KAboutContributor

KAboutContributor창문부품은 About칸에 현시할 개별적인 공헌자의 이름과 기타 정보를 표준형식으로 제시한다.

머리부파일

```
#include <kaboutdialog.h>
```

기초클라스

QFrame QObject QPaintDevice QWidget Qt

구성자

```
KAboutContributor(QWidget *parent = 0, const char *name = 0, const QString &username = QString::null,
    const QString &email = QString::null, const QString &url = QString::null,
    const QString &work = QString::null, bool showHeader = false, bool showFrame = true,
    bool showBold = false);
```

메쏘드

```
QString getEmail(void);
QString getName(void);
QString getURL(void);
QString getWork(void);
void setEmail(const QString &text, const QString &header = QString::null, bool update = true);
void setName(const QString &text, const QString &header = QString::null, bool update = true);
void setURL(const QString &text, const QString &header = QString::null, bool update = true);
void setWork(const QString &text, const QString &header = QString::null, bool update = true);
virtual QSize sizeHint(void) const;
```

신호

```
void openURL(const QString &url);
void sendEmail(const QString &name, const QString &email);
```

다음의 실례는 그림 19-2에 보여주는 KAboutContributor창문부품을 만들고 현시한다.

```
/* showaboutcontributor.cpp */
#include <kapplication.h>
#include <kaboutdialog.h>
int main(int argc,char **argv)
```

```
{
    KApplication app(argc,argv, "aboutcontributor");
    KAboutContributor *aboutcontributor = new KAboutContributor(0,0, "Phillip Space",
        "phil@nobody.com","http://www.belugalake.com", "Responsible for coloring pixels");
    aboutcontributor->show();
    app.setMainWidget(aboutcontributor);
    return(app.exec());
}
```
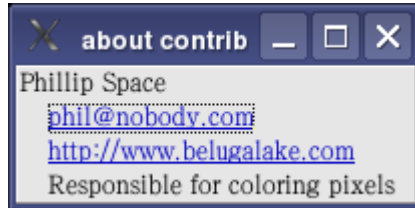
그림 19-2. 개인정보를 현시하는 KAboutContributor창문부품

KAboutDialog

KAboutDialog창문부품은 응용프로그람에 대한 정보를 현시하는 튀여나오기대화칸이다.

머리부파일

```
#include <kaboutdialog.h>
```

기초클라스

KDialog KDialogBase QDialog QObject QPaintDevice QWidget Qt

파생클라스

KAboutKDE

구성자

```
KAboutDialog(QWidget *parent = 0, const char *name = 0, bool modal = true);
KAboutDialog(int dialogLayout, const QString &caption, int buttonMask, ButtonCode defaultButton,
    QWidget *parent = 0, const char *name = 0, bool modal = false, bool separator = false,
    const QString &user1 = QString::null, const QString &user2 = QString::null,
    const QString &user3 = QString::null);
```

메쏘드

```
KAboutContainer *addContainer(int childAlignment, int innerAlignment);
KAboutContainer *addContainerPage(const QString &title, int childAlignment = AlignCenter,
        int innerAlignment = AlignCenter);
void addContributor(const QString &name, const QString &email, const QString &url, const QString
&work);
```

```
QFrame *addPage(const QString &title);
QFrame *addTextPage(const QString &title, const QString &text, bool richText = false,
        int numLines = 10);
void adjust();
static void imageURL(QWidget *parent, const QString &caption, const QString &path,
        const QColor &imageColor, const QString &url);
void setAuthor(const QString &name, const QString &email, const QString &url, const QString &wor
k);
void setImage(const QString &fileName);
void setImageBackgroundColor(const QColor &color);
void setImageFrame(bool state);
void setLogo(const QPixmap &);
void setMaintainer(const QString &name, const QString &email, const QString &url, const QString &
work);
void setProduct(const QString &appName, const QString &version, const QString &author,
        const QString &year);
void setTitle(const QString &title);
void setVersion(const QString &name);
virtual void show(void);
virtual void show(QWidget *centerParent);
```

신호

```
void openURL(const QString &url);
void sendEmail(const QString &name, const QString &email);
```

렬거형

```
enum LayoutType { AbtPlain=0x0001, AbtTabbed=0x0002, AbtTitle=0x0004, AbtImageLeft=0x0008,
    AbtImageRight=0x0010, AbtImageOnly=0x0020, AbtProduct=0x0040,
    AbtKDEStandard=AbtTabbed|AbtTitle|AbtImageLeft, AbtAppStandard=AbtTabbed|AbtTitle|AbtProduct,
    AbtImageAndTitle=AbtPlain|AbtTitle|AbtImageOnly };
```

5장에 KAboutDialog의 실례가 있다.

KAboutKDE

KAboutKDE대화칸은 KAboutDialog에 기초하며 표준KDE형식으로 현시하도록 구성된다.

머리부파일

```
#include <kaboutkde.h>
```

기초클라스

KAboutDialog KDialog KDialogBase QDialog QObject

QPaintDevice QWidget Qt

구성자

```
KAboutKDE(QWidget *parent = 0, const char *name = 0, bool modal = true);
```

KAboutDialog의 KDE표준형식을 보여주는 실례는 5장에 있다.

KAboutWidget

KAboutWidget는 여러 형식들중 하나로 응용프로그람정보를 현시하며 현시정보를 삽입하는데 쓰이는 메쏘드모임을 제공한다. 이것은 KAboutDialog의 기본창문부품이다.

머리부파일

```
#include <kaboutdialog.h>
```

기초클라스

QObject QPaintDevice QWidget Qt

구성자

```
KAboutWidget(QWidget *parent = 0, const char *name = 0);
```

메쏘드

```
void addContributor(const QString &name, const QString &email, const QString &url,
        const QString &work);
void adjust();
void setAuthor(const QString &name, const QString &email, const QString &url,
        const QString &work);
void setLogo(const QPixmap &);
void setMaintainer(const QString &name, const QString &email, const QString &url,
        const QString &work);
void setVersion(const QString &name);
```

신호

```
void openURL(const QString &url);
void sendEmail(const QString &name, const QString &email);
```

5장에 KAboutWidget를 KAboutDialog의 기본창문부품으로 사용하는 실례가 있다.

KAccelMenu

KAccelMenu창문부품은 KAccel과 KKeyDialog의 사용을 간단화하는 파생클라스이다.

머리부파일

```
#include <kaccelmenu.h>
```

기초클라스

QFrame QMenuData QObject QPaintDevice QPopupMenu QWidget Qt

구성자

KAccelMenu(KAccel *k, QWidget *parent = 0, const char *name = 0);

메쏘드

int insItem(const QPixmap &pixmap, const char *text, const char *action, const QObject *receiver,
    const char *member, const char *accel = 0);

int insItem(const char *text, const char *action, const QObject *receiver, const char *member,
    const char *accel = 0);

int insItem(const QPixmap &pixmap, const char *text, const char *action, const QObject *receiver,
    const char *member, KStdAccel::StdAccel accel);

int insItem(const char *text, const char *action, const QObject *receiver, const char *member,
    KStdAccel::StdAccel accel);

KAnimWidget

KAnimWidget는 개별적인 픽스매프들을 차례로 현시함으로써 동화를 생성한다.

머리부파일

#include <kanimwidget.h>

기초클라스

QFrame QObject QPaintDevice QWidget Qt

구성자

KAnimWidget(const QStringList &icons, int size = 0, QWidget *parent = 0L, const char *name = 0L);

KAnimWidget(QWidget *parent = 0L, const char *name = 0L);

메쏘드

void setIcons(const QStringList &icons);

void setSize(int size);

void start();

void stop();

신호

void clicked();

다음의 실례는 그림 19-3에 보여진 창문을 현시하고 flag1.png로부터 flag4.png라는 이름의 픽스매프들을 적재하여 프레임들의 동화렬로서 현시한다. share/icons/small등록부들에서 그림기호파일들을 탐색하고 pics등록부에서 응용프로그람을 찾는다. 이 실례는 ~/.kde/animwidget/pics에서 파일들을 찾는다.

/* showanimwidget.cpp */

```
#include <kapplication.h>
#include <kanimwidget.h>
#include <qstringlist.h>
int main(int argc,char **argv)
{
   KApplication app(argc,argv, "animwidget");
   QStringList icons;
   QWidget *widget = new QWidget();
   icons.append("flag1");
   icons.append("flag2");
   icons.append("flag3");
   icons.append("flag4");
   KAnimWidget *kanimwidget = new KAnimWidget(icons,0,widget);
   kanimwidget->start();
   widget->show();
   app.setMainWidget(widget);
   return(app.exec());
}
```

그림 19-3. KAnimWidget가 관리하는 동화렬의 한개 프레임

KAuthIcon

KAuthIcon창문부품은 사용자가 어떤 작용을 수행할 능력을 가지는가를 가리키도록 설계된 창문부품들의 기초클라스이다.

머리부파일

```
#include <kauthicon.h>
```

기초클라스

QObject QPaintDevice QWidget Qt

파생클라스

KRootPermsIcon KWritePermsIcon

구성자

```
KAuthIcon(QWidget *parent = 0, const char *name = 0);
```

메쏘드

virtual QSize sizeHint() const;

virtual bool status() const = 0;

처리부

virtual void updateStatus() = 0;

신호

void authChanged(bool authorized);

KBugReport

KBugReport는 사용자로부터 오유보고를 받아들이는 대화칸의 기초클라스이다. KHelpMenu가 KBugReport대화칸을 만들므로 이것들중의 하나를 창조할 필요는 없다.

머리부파일

#include <kbugreport.h>

기초클라스

KDialog KDialogBase QDialog QObject QPaintDevice QWidget Qt

구성자

KBugReport(QWidget *parent = 0L, bool modal = true, const KAboutData *aboutData = 0L);

KButtonBox

KButtonBox는 수직으로 또는 수평으로 조직화된 단추그룹을 보유하는 용기이다.

머리부파일

#include <kbuttonbox.h>

기초클라스

QObject QPaintDevice QWidget Qt

구성자

KButtonBox(QWidget *parent, int _orientation = HORIZONTAL, int border = 0, int _autoborder = 6);

메쏘드

QPushButton *addButton(const QString &text, bool noexpand = FALSE);

void addStretch(int scale = 1);

void layout();

virtual void resizeEvent(QResizeEvent *);

virtual QSize sizeHint() const;

렬거형

enum (anon) { VERTICAL=1, HORIZONTAL=2 };

7장에 KButtonBox의 실례가 있다.

KCharSelect

KCharSelect창문부품은 사용자가 서체를 선택하고 서체로부터 하나의 문자를 선택한다.

머리부파일

```
#include <kcharselect.h>
```

기초클라스

QFrame QHBox QObject QPaintDevice QVBox QWidget Qt

구성자

```
KCharSelect(QWidget *parent, const char *name, const QString &font = QString::null,
        const QChar &chr = , int tableNum = 0);
```

메쏘드

```
virtual QChar chr();
virtual void enableFontCombo(bool e);
virtual void enableTableSpinBox(bool e);
virtual QString font();
virtual bool isFontComboEnabled();
virtual bool isTableSpinBoxEnabled();
virtual void setChar(const QChar &chr);
virtual void setFont(const QString &font);
virtual void setTableNum(int tableNum);
virtual QSize sizeHint() const;
virtual int tableNum();
```

신호

```
void activated(const QChar &c);
void activated();
void focusItemChanged();
void focusItemChanged(const QChar &c);
void fontChanged(const QString &_font);
void highlighted(const QChar &c);
void highlighted();
```

다음의 실례는 그림 19-4에 보여주는 KCharSelect창문을 현시한다.

```
/* showcharselect.cpp */
#include <kapplication.h>
#include <kcharselect.h>
#include <qstring.h>
int main(int argc,char **argv)
```

```
{
    KApplication app(argc,argv, "showcharselect");
    QWidget *widget = new QWidget();
    KCharSelect *kcharselect = new KCharSelect(widget, "charselect");
    kcharselect->resize(kcharselect->sizeHint());
    widget->resize(kcharselect->sizeHint());
    widget->show();
    app.setMainWidget(widget);
    return(app.exec());
}
```
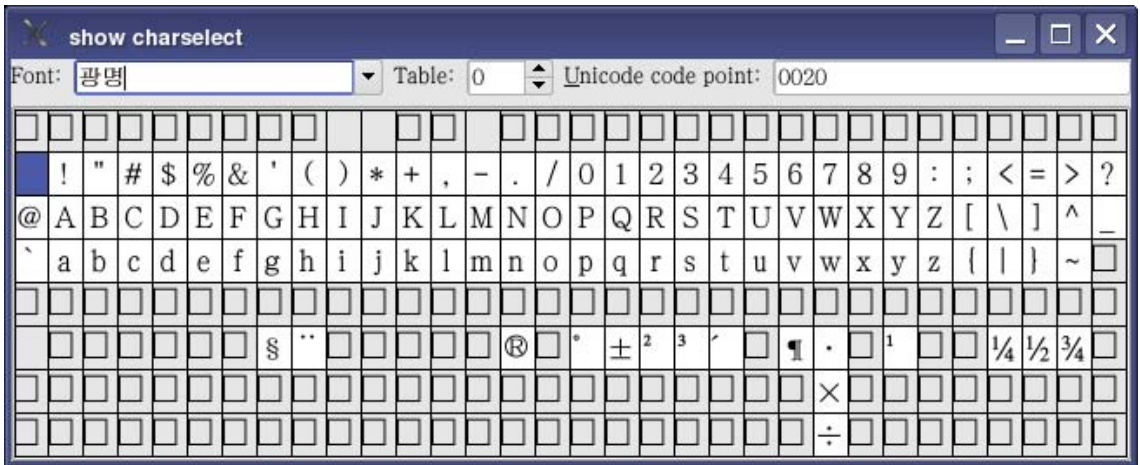
그림 19-4. 하나의 서체와 문자를 선택한 KCharSelect창문부품

KCharSelectTable

KCharSelectTable창문부품은 KCharSelect창문부품의 문자현시부분이다.

머리부파일

```
#include <kcharselect.h>
```

기초클라스

QFrame QObject QPaintDevice QTableView QWidget Qt

구성자

```
KCharSelectTable(QWidget *parent, const char *name, const QString &_font, const QChar &_chr,
        int _tableNum);
```

메쏘드

```
virtual QChar chr();
virtual void setChar(const QChar &_chr);
virtual void setFont(const QString &_font);
```

```
virtual void setTableNum(int _tableNum);
virtual QSize sizeHint() const;
```

신호

```
void activated(const QChar &c);
void activated();
void focusItemChanged();
void focusItemChanged(const QChar &c);
void highlighted(const QChar &c);
void highlighted();
void tableDown();
void tableUp();
```

실례로 KCharSelect를 보시오.

KCModule

KCModule창문부품은 모든 조종모듈들의 기초클라스이다. 결과의 창문부품은 kcontrol창문중의 하나로 나타나며 환경설정값을 조정하는데 쓰인다.

머리부파일

```
#include <kcmodule.h>
```

기초클라스

QObject QPaintDevice QWidget Qt

구성자

```
KCModule(QWidget *parent = 0, const char *name = 0)
 : QWidget( parent , name ) , _btn ( Help | Default | Reset | Cancel | Apply | Ok );
```

메쏘드

```
int buttons();
virtual void defaults();
static void init();
virtual void load();
virtual QString quickHelp();
virtual void save();
virtual void sysdefaults();
```

신호

```
void changed(bool state);
```

렬거형

```
enum Button { Help=1, Default=2, Reset=4, Cancel=8, Apply=16, Ok=32, SysDefault=64 };
```

KColorButton

KColorButton창문부품은 색칠된 누름단추이며 선택하였을 때 사용자가 다른 색을 선택할수 있게 하는 대화칸을 펼친다.

머리부파일

```
#include <kcolorbtn.h>
```

기초클라스

QButton QObject QPaintDevice QPushButton QWidget Qt

구성자

```
KColorButton(QWidget *parent, const char *name = 0L);
KColorButton(const QColor &c, QWidget *parent, const char *name = 0L);
```

메쏘드

```
const QColor color() const;
void setColor(const QColor &c);
```

신호

```
void changed(const QColor &newColor);
```

다음의 실례는 붉은색의 KColorButton을 만든다. 그림 19-5에 보여준것처럼 단추를 선택하면 대화칸이 나나난다. 그다음 사용자가 다른 색을 선택하면 KColorButton의 색을 변경하고 changed()신호를 발생시킨다.

```
/* showcolorbutton.cpp */
#include <kapplication.h>
#include <kcolorbtn.h>
#include <qcolor.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showcolorbutton");
    KColorButton *colorbutton = new KColorButton(0, "colorbutton");
    colorbutton->setColor(QColor("red"));
    colorbutton->resize(colorbutton->sizeHint());
    colorbutton->show();
    app.setMainWidget(colorbutton);
    return(app.exec());
}
```
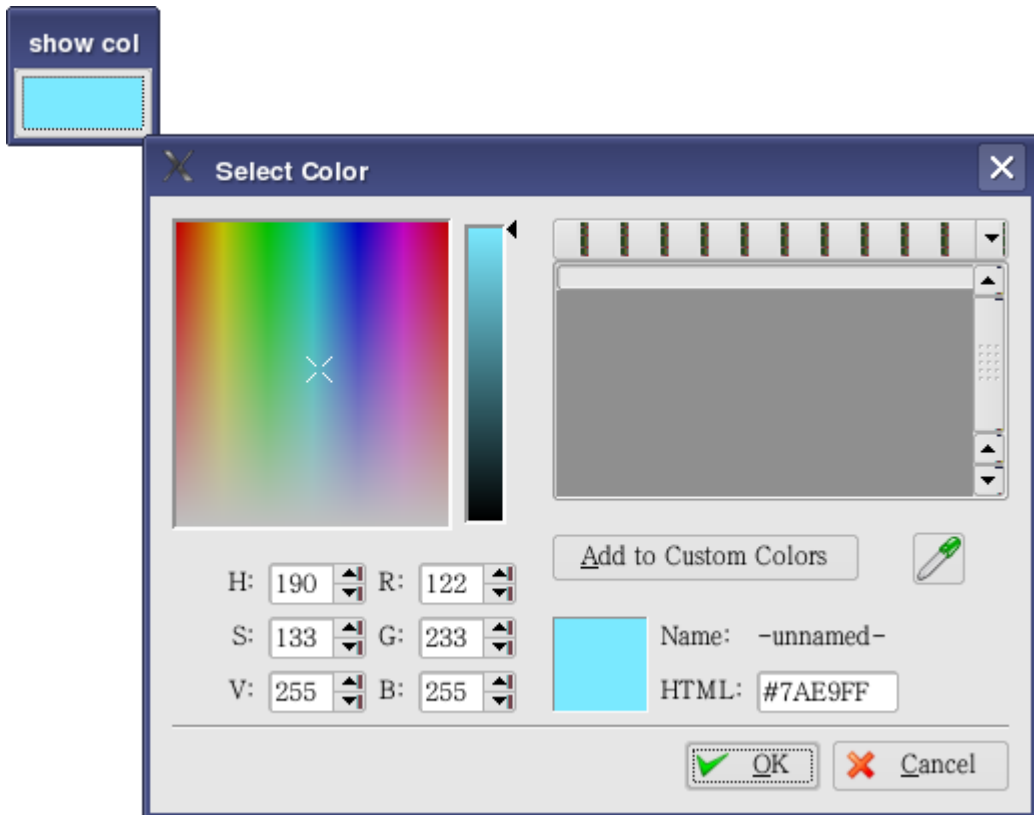
그림 19-5. 색선택대화칸을 펼치는 KColorButton

KColorCells

KColorCells창문부품은 색모임을 현시하고 사용자가 그것들중 하나를 선택하게 한다.

머리부파일

```
#include <kcolordlg.h>
```

기초클라스

QFrame QObject QPaintDevice QTableView QWidget Qt

구성자

```
KColorCells(QWidget *parent, int rows, int cols);
```

메쏘드

```
QColor color(int indx);
int getSelected();
int numCells();
void setAcceptDrags(bool _acceptDrags);
void setColor(int colNum, const QColor &col);
void setShading(bool _shade);
```

신호

```
    void colorSelected(int col);
```

다음의 실례는 그림 19-6에 보여주는 색세포목록을 현시한다.

```
/* showcolorcells.cpp */
#include <kapplication.h>
#include <kcolordlg.h>
int main(int argc,char **argv)
{
   KApplication app(argc,argv, "showcolorcells");
   KColorCells *colorcells = new KColorCells(0,1,5);
   colorcells->setColor(0,QColor("magenta"));
   colorcells->setColor(1,QColor("red"));
   colorcells->setColor(2,QColor("blue"));
   colorcells->setColor(3,QColor("green"));
   colorcells->setColor(4,QColor("cyan"));
   colorcells->show();
   app.setMainWidget(colorcells);
   return(app.exec());
}
```

그림 19-6. 5가지 색들의 렬을 보여주는 KColorCells창문부품

KColorCombo

KColorCombo창문부품은 마우스로 선택하기 위한 내리펼침형의 색목록을 현시한다.

머리부파일

```
    #include <kcolordlg.h>
```

기초클라스

    QComboBox QObject QPaintDevice QWidget Qt

구성자

```
    KColorCombo(QWidget *parent, const char *name = 0L);
```

메쏘드

```
    void setColor(const QColor &col);
```

처리부

```
void slotActivated(int index);
void slotHighlighted(int index);
```

신호

```
void activated(const QColor &col);
void highlighted(const QColor &col);
```

다음의 실례는 KColorCombo창문부품을 현시한다. 그림 19-7은 KColorCombo창문부품이 펼쳐지고 색이 선택된 후의 상태를 보여준다.

```
/* showcolorcombo.cpp */
#include <kapplication.h>
#include <kcolordlg.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showcolorcombo");
    KColorCombo *colorcombo = new KColorCombo(0, "colorcombo");
    colorcombo->show();
    app.setMainWidget(colorcombo);
    return(app.exec());
}
```

그림 19-7. 목록으로부터 색을 선택할수 있는 KColorCombo창문부품

KColorDialog

KColorDialog은 사용자정의색을 비롯한 여러가지 특성을 가지는 색선택대화칸이다.

머리부파일

```
#include <kcolordlg.h>
```

기초클라스

KDialog KDialogBase QDialog QObject QPaintDevice QWidget Qt

구성자

```
KColorDialog(QWidget *parent = 0L, const char *name = 0L, bool modal = FALSE);
```

메쏘드

```
QColor color();
static int getColor(QColor &theColor, QWidget *parent = 0L);
static QColor grabColor(const QPoint &p);
```

처리부

```
void setColor(const QColor &col);
```

신호

```
void colorSelected(const QColor &col);
```

11장에 KColorDialog창문부품의 실례가 있다.

KColorPatch

KColorPatch창문부품은 색의 직4각형구역을 현시하고 마우스선택에 응답한다.

머리부파일

```
#include <kcolordlg.h>
```

기초클라스

```
QFrame QObject QPaintDevice QWidget Qt
```

구성자

```
KColorPatch(QWidget *parent);
```

메쏘드

```
void setColor(const QColor &col);
```

신호

```
void colorChanged(const QColor &);
```

다음의 실례는 그림 19-8에 보여주는것처럼 색령역을 현시하기 위하여 KColorPatch창문부품을 제일웃준위창문으로 사용한다.

```
/* showcolorpatch.cpp */
#include <kapplication.h>
#include <kcolordlg.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showcolorpatch");
    KColorPatch *colorpatch = new KColorPatch(0);
    colorpatch->setColor(QColor("blue"));
    colorpatch->show();
    app.setMainWidget(colorpatch);
    return(app.exec());
}
```
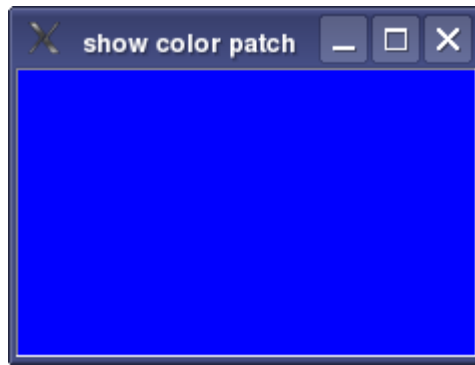
그림 19-8. 색령역을 보여주는 KColorPatch

KComboBox

KComboBox창문부품은 사용자가 목록으로부터 선택할수 있는 단추로서 늘 웃끝에 현재 항목을 현시한다.

머리부파일

<include <kcombobox.h>

기초클라스

KCompletionBase QComboBox QObject QPaintDevice QWidget Qt

파생클라스

KFileComboBox KFileFilter KURLComboBox

구성자

KComboBox(QWidget *parent = 0, const char *name = 0);

KComboBox(bool rw, QWidget *parent = 0, const char *name = 0);

메쏘드

bool autoCompletion() const;

int cursorPosition() const;

bool isContextMenuEnabled() const;

bool isEditable() const;

virtual void setAutoCompletion(bool autocomplete);

virtual void setEnableContextMenu(bool showMenu);

처리부

void rotateText(KeyBindingType);

신호

void completion(const QString &);

void nextMatch(KeyBindingType);

void previousMatch(KeyBindingType);

```
void returnPressed();
void returnPressed(const QString &);
void rotateDown(KeyBindingType);
void rotateUp(KeyBindingType);
```

다음의 실례는 4개 항목을 가지는 KComboBox를 만든다. 그림 19-9는 마우스로 목록을 펼친 후 기정으로 3번째 항목을 선택한 창문부품을 보여준다.

```
/* showcombobox.cpp */
#include <kapplication.h>
#include <kcombobox.h>
const char *list[] = { "First Selection", "Second Selection", "Third Selection", "Fourth Selection" };
int main(int argc,char **argv)
{
   KApplication app(argc,argv, "showcombobox");
   KComboBox *combobox = new KComboBox();
   combobox->insertStrList(list,4);
   combobox->show();
   app.setMainWidget(combobox);
   return(app.exec());
}
```

그림 19-9. 3번째 항목을 선택한 KComboBox

KDatePicker

KDatePicker는 사용자에게 달력을 현시하고 마우스로 날자를 선택할수 있게 한다.

머리부파일

```
#include <kdatepik.h>
```

기초클라스

QFrame QObject QPaintDevice QWidget Qt

구성자

```
KDatePicker(QWidget *parent = 0, QDate = QDate::currentDate (), const char * name = 0 );
```

메쏘드

```
const QDate & getDate();
bool setDate(const QDate &);
```

```
void  setEnabled(bool);
void  setFontSize(int);
QSize  sizeHint()  const;
```

신호

```
void  dateChanged(QDate);
void  dateEntered(QDate);
void  dateSelected(QDate);
void tableClicked();
```

다음의 실례는 제일 웃준위창문으로서 KDatePicker창문부품을 현시한다. 그림 19-10에 보여준것처럼 웃끝의 왼쪽과 오른쪽 화살표들은 월과 년을 변경하는데 사용할수 있다. 날자를 달력으로부터 선택한다. KDatePicker창문크기가 변경되면 달력과 매개 부분의 크기도 변경되므로 전체는 그 형태를 유지한다.

```
/*  showdatepicker.cpp  */
#include  <kapplication.h>
#include  <kdatepik.h>
int  main(int  argc,char  **argv)
{
    KApplication  app(argc,argv,  "showdatepicker");
    KDatePicker  *datepicker  =  new  KDatePicker();
    datepicker->resize(datepicker->sizeHint());
    datepicker->show();
    app.setMainWidget(datepicker);
    return(app.exec());
}
```

그림 19-10. KDatePicker창문부품

KDateTable

KDateTable은 한달을 현시하고 사용자가 마우스로 날자를 선택할수 있게 한다.

머리부파일

#include <kdatetbl.h>

기초클라스

QFrame QObject QPaintDevice QTableView QWidget Qt

구성자

KDateTable(QWidget *parent = 0, QDate date = QDate::currentDate () , const char * name = 0 , WFlags f = 0);

메쏘드

const QDate & getDate();

bool setDate(const QDate &);

void setFontSize(int size);

QSize sizeHint() const;

신호

void dateChanged(QDate);

void tableClicked();

KDateTable창문부품은 독립형 날자선택기가 아니라 KDatePicker와 같은 다른 창문부품의 구성요소이다. 그림 19-11에 보여주는것처럼 월이름이 현시되지 않는다.

```
/* showdatetable.cpp */
#include <kapplication.h>
#include <kdatetbl.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showdatetable");
    KDateTable *datetable = new KDateTable();
    datetable->show();
    app.setMainWidget(datetable);
    return(app.exec());
}
```
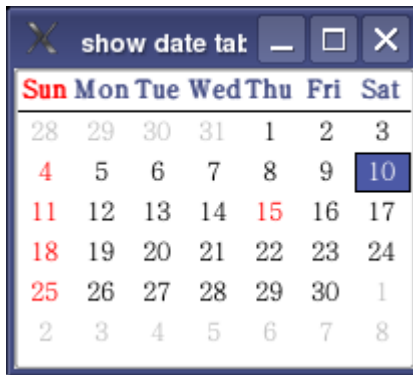
그림 19-11. KDateTable창문부품

KDialog

KDialog창문부품은 이행허용KDE창문부품들의 기초클라스로서 QDialog를 확장하여 대화칸용 표준KDE메쏘드모임을 포함한다. 이것은 KDialogBase가 구축되는 기초클라스이다.

머리부파일

#include <kdialog.h>

기초클라스

QDialog QObject QPaintDevice QWidget Qt

파생클라스

KAboutDialog KAboutKDE KBugReport KColorDialog KCookieWin KDialogBase KEdFind KEdGotoLine

KEdReplace KEditToolbar KFileDialog KFileDialogConfigureDlg KFontDialog KIconDialog KKeyDialog

KLineEditDlg KPasswordDialog KTextPrintDialog KURLRequesterDlg KabAPI

구성자

```
KDialog(QWidget *parent = 0, const char *name = 0, bool modal = false, WFlags f = 0);
```

메쏘드

```
static int marginHint();
static void resizeLayout(QWidget *widget, int margin, int spacing);
static void resizeLayout(QLayoutItem *lay, int margin, int spacing);
static int spacingHint();
```

처리부

```
virtual void setCaption(const QString &caption);
virtual void setPlainCaption(const QString &caption);
```

신호

```
void layoutHintChanged();
```

KDialogBase

KDialogBase는 KDE에서 대화칸을 만드는 기초클라스로서 대화칸용의 표준KDE단추모임을 포함한다.

머리부파일

#include <kdialogbase.h>

기초클라스

KDialog QDialog QObject QPaintDevice QWidget Qt

파생클라스

KAboutDialog KAboutKDE KBugReport KColorDialog KCookieWin KEdFind KEdGotoLine KEdReplace

KEditToolbar KFileDialog KFileDialogConfigureDlg KFontDialog KIconDialog KKeyDialog KLineEditDlg

KPasswordDialog KTextPrintDialog KURLRequesterDlg KabAPI

구성자

```
KDialogBase(QWidget *parent = 0, const char *name = 0, bool modal = true,
    const QString &caption = QString::null, int buttonMask = Ok | Apply | Cancel,
    ButtonCode defaultButton = Ok, bool separator = false, const QString &user1 = QString::null,
    const QString &user2 = QString::null, const QString &user3 = QString::null);
KDialogBase(int dialogFace, const QString &caption, int buttonMask, ButtonCode defaultButton,
    QWidget *parent = 0, const char *name = 0, bool modal = true, bool separator = false,
    const QString &user1 = QString::null, const QString &user2 = QString::null,
    const QString &user3 = QString::null);
KDialogBase(const QString &caption, int buttonMask = Yes | No | Cancel,
    ButtonCode defaultButton = Yes, ButtonCode escapeButton = Cancel, QWidget *parent = 0,
    const char *name = 0, bool modal = true, bool separator = false, QString yes = QString::null,
    QString no = QString::null, QString cancel = QString::null);
```

메쏘드

```
QPushButton *actionButton(ButtonCode id);
int activePageIndex() const;
QGrid *addGridPage(int n, QGrid::Direction dir, const QString &itemName,
    const QString &header = QString::null, const QPixmap &pixmap = QPixmap () );
QHBox *addHBoxPage(const QString &itemName, const QString &header = QString::null,
    const QPixmap &pixmap = QPixmap () );
QFrame *addPage(const QString &item, const QString &header = QString::null,
    const QPixmap &pixmap = QPixmap () );
```

```
QVBox *addVBoxPage(const QString &itemName, const QString &header = QString::null,
        const QPixmap &pixmap = QPixmap () );
virtual void adjustSize();
QSize calculateSize(int w, int h);
void delayedDestruct();
void disableResize();
void enableButtonSeparator(bool state);
static const QPixmap *getBackgroundTile();
void getBorderWidths(int &ulx, int &uly, int &lrx, int &lry) const;
QRect getContentsRect();
QWidget *getMainWidget();
static bool haveBackgroundTile();
QString helpLinkText();
void incInitialSize(const QSize &s, bool noResize = false);
QGrid *makeGridMainWidget(int n, QGrid::Direction dir);
QHBox *makeHBoxMainWidget();
QFrame *makeMainWidget();
QVBox *makeVBoxMainWidget();
int pageIndex(QWidget *widget) const;
QFrame *plainPage();
static void setBackgroundTile(const QPixmap *pix);
void setButtonApplyText(const QString &text = QString::null, const QString &tooltip = QString::null,
        const QString &quickhelp = QString::null);
void setButtonCancelText(const QString &text = QString::null, const QString &tooltip = QString::null,
        const QString &quickhelp = QString::null);
void setButtonOKText(const QString &text = QString::null, const QString &tooltip = QString::null,
        const QString &quickhelp = QString::null);
void setButtonText(ButtonCode id, const QString &text);
void setButtonTip(ButtonCode id, const QString &text);
void setButtonWhatsThis(ButtonCode id, const QString &text);
void setIconListAllVisible(bool state);
void setInitialSize(const QSize &s, bool noResize = false);
void setMainWidget(QWidget *widget);
void setTreeListAutoResize(bool state);
void showButton(ButtonCode id, bool state);
void showButtonApply(bool state);
```

void showButtonCancel(bool state);

void showButtonOK(bool state);

bool showPage(int index);

void showTile(bool state);

처리부

void enableButton(ButtonCode id, bool state);

void enableButtonApply(bool state);

void enableButtonCancel(bool state);

void enableButtonOK(bool state);

void enableLinkedHelp(bool state);

void helpClickedSlot(const QString &);

void setHelp(const QString &path, const QString &topic);

void setHelpLinkText(const QString &text);

void updateBackground();

신호

void apply();

void applyClicked();

void backgroundChanged();

void cancelClicked();

void closeClicked();

void defaultClicked();

void helpClicked();

void hidden();

void noClicked();

void okClicked();

void tryClicked();

void user1Clicked();

void user2Clicked();

void user3Clicked();

void yesClicked();

렬거형

enum ButtonCode { Help=0x00000001, Default=0x00000002, Ok=0x00000004, Apply=0x00000008,

Try=0x00000010, Cancel=0x00000020, Close=0x00000040, User1=0x00000080,

User2=0x00000100, User3=0x00000200, No=0x00000080,

Yes=0x00000100, Stretch=0x80000000 };

enum ActionButtonStyle { ActionStyle0=0, ActionStyle1, ActionStyle2, ActionStyle3, ActionStyle4,

```
        ActionStyleMAX };
    enum DialogType { TreeList=KJanusWidget::TreeList, Tabbed=KJanusWidget::Tabbed,
        Plain=KJanusWidget::Plain, Swallow=KJanusWidget::Swallow,
        IconList=KJanusWidget::IconList };
```

KDialogBase를 사용하여 대화칸을 구축하는 실례들이 4장에 있다

KDialogBaseButton

KDialogBaseButton은 KDialogBase가 내부적으로 사용하며 역호출메쏘드들에서 쓰이는 단추의 유일식별번호들을 추가한다.

머리부파일

```
#include <kdialogbase.h>
```

기초클라스

QButton QObject QPaintDevice QPushButton QWidget Qt

구성자

```
KDialogBaseButton(const QString &text, int key,
QWidget *parent = 0, const char *name = 0);
```

메쏘드

```
inline int id();
```

4장에 KDialogBase를 사용하여 대화칸을 구축하는 실례들이 있다.

KDockMainWindow

KDockMainWindow는 KDockWidget들이 경계를 따라 류동하게 하는 KMainWindow의 특수판이다.

머리부파일

```
#include <kdockwidget.h>
```

기초클라스

KMainWindow KXMLGUIBuilder KXMLGUIClient QObject QPaintDevice QWidget Qt

구성자

```
KDockMainWindow(const char *name = 0L);
```

메쏘드

```
void activateDock();
KDockWidget *createDockWidget(const QString &name, const QPixmap &pixmap,
    QWidget *parent = 0L);
QPopupMenu *dockHideShowMenu();
KDockWidget *getMainDockWidget();
```

```
void makeDockInvisible(KDockWidget *dock);
void makeDockVisible(KDockWidget *dock);
void makeWidgetDockVisible(QWidget *widget);
KDockManager *manager();
void readDockConfig(KConfig *c = 0L, QString group = QString::null);
void setMainDockWidget(KDockWidget *);
void setCentralWidget(QWidget *);
void writeDockConfig(KConfig *c = 0L, QString group = QString::null);
```

다음의 실례는 그림 19-12에 보여주는 제일 웃준위창문부품으로서 KDockMainWindow를 사용한다.

```
/* showdockmainwindow.cpp */
#include <kapplication.h>
#include <kdockwidget.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showdatetable");
    KDockMainWindow *dockmainwindow = new KDockMainWindow();
    dockmainwindow->setBackgroundColor(QColor("blue"));
    dockmainwindow->show();
    app.setMainWidget(dockmainwindow);
    return(app.exec());
}
```
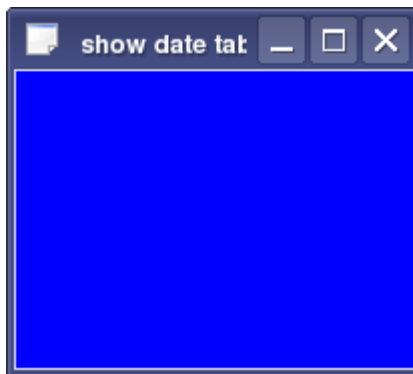
그림 19-12. 제일 웃준위창문으로서 사용하는 KDockMainWindow

류동가능창문부품들을 추가하는 실례는 KDockWidget를 보시오.

KDockWidget

KDockWidget는 KDockMainWindow창문부품안의 류동가능창문부품들중 하나의 창문부품

을 보유하는 특수한 용기창문부품이다.

머리부파일

```
#include <kdockwidget.h>
```

기초클라스

QObject QPaintDevice QWidget Qt

구성자

```
KDockWidget(KDockManager *dockManager, const char *name, const QPixmap &pixmap,
    QWidget *parent = 0L);
```

메쏘드

```
KDockManager *dockManager();
int dockSite();
int enableDocking();
virtual bool event(QEvent *);
QWidget *getWidget();
bool isDockBackPossible();
void makeDockVisible();
KDockWidget *manualDock(KDockWidget *target, DockPosition dockPos, int spliPos = 50,
    QPoint pos = QPoint ( 0) , bool check = false );
bool mayBeHide();
bool mayBeShow();
void setDockSite(int pos);
void setEnableDocking(int pos);
void setHeader(KDockWidgetAbstractHeader *ah);
void setToolTipString(const QString &ttStr);
void setWidget(QWidget *w);
virtual void show();
const QString & toolTipString();
```

처리부

```
void changeHideShowState();
void dockBack();
void undock();
```

신호

```
void docking(KDockWidget *dw, KDockWidget::DockPosition dp);
void headerCloseButtonClicked();
void headerDockbackButtonClicked();
void iMBeingClosed();
```

```
void setDockDefaultPos();
```

렬거형

```
enum DockPosition { DockNone=0, DockTop=0x0001, DockLeft=0x0002, DockRight=0x0004,
        DockBottom=0x0008, DockCenter=0x0010, DockDesktop=0x0020,
        DockCorner=DockTop|DockLeft|DockRight|DockBottom,
        DockFullSite=DockCorner|DockCenter, DockFullDocking=DockFullSite|DockDesktop };
```

다음의 실례는 3개의 KDockWidget창문부품을 사용하여 KDockMainWindow에 3개 창문부품을 추가한다. 그림 19-13에 보여주는것처럼 창문부품들은 조종판창문에 현시된다. 전체창문은 마우스에 의하여 구성될수 있다. 또한 창문크기를 늘이고 줄이는데 쓰이는 창문부품들사이의 띠들뿐아니라 매개 류동된 창문부품들의 출현을 허용 또는 금지하는데 쓰이는 조종띠(류동가능)가 있다. setMainDockWidget()호출은 그 경계들을 따라 류동된 창문부품들에 의해 로출된 공간에 나타나는 창문부품을 지정하며 이 실례에서 그것은 오른쪽웃구석의 구역이다.

```
/* showdockwidget.cpp */
#include <kapplication.h>
#include <kdockwidget.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showdatetable");
    KDockMainWindow *dockmainwindow = new KDockMainWindow();
    QPixmap pixmap("idea.png");
    KDockWidget *mainDock = dockmainwindow->createDockWidget("Main Dock",pixmap);
    QWidget *actualMain = new QWidget(mainDock);
    actualMain->setBackgroundColor(QColor("green"));
    actualMain->setMinimumSize(200,200);
    dockmainwindow->setCentralWidget(mainDock);
    dockmainwindow->setMainDockWidget(mainDock);
    KDockWidget *leftDock = dockmainwindow->createDockWidget("Left Dock",pixmap);
    QWidget *actualLeft = new QWidget(leftDock);
    actualLeft->setBackgroundColor(QColor("blue"));
    actualLeft->setMinimumSize(200,200);
    leftDock->manualDock(mainDock, KDockWidget::DockLeft,20);
    KDockWidget *bottomDock = dockmainwindow->createDockWidget("Bottom Dock",pixmap);
    QWidget *actualBottom = new QWidget(bottomDock);
    actualBottom->setBackgroundColor(QColor("red"));
    actualBottom->setMinimumSize(200,200);
```

```
    bottomDock->manualDock(mainDock, KDockWidget::DockBottom,20);
    dockmainwindow->activateDock();
    app.setMainWidget(dockmainwindow);
    return(app.exec());
}
```
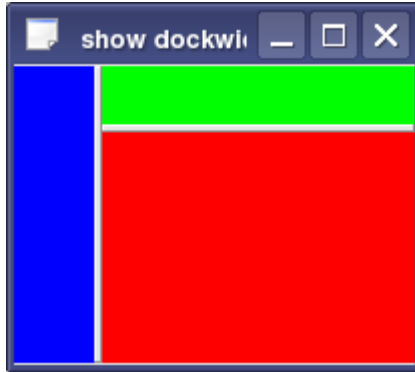
그림 19-13. KDockMainWindow안의 3개 창문부품을 류동하는데 사용한 KDockWidget

KDoubleNumInput

KDoubleNumInput창문부품은 류점수들을 입력하고 사용자입력을 확인한다.

머리부파일

```
#include <knuminput.h>
```

기초클라스

KNumInput QObject QPaintDevice QWidget Qt

구성자

```
KDoubleNumInput(double value, QWidget *parent = 0, const char *name = 0);
KDoubleNumInput(KNumInput *below, double value, QWidget *parent = 0, const char *name = 0);
```

메쏘드

```
virtual QSize minimumSizeHint() const;
void setFormat(const char *format);
virtual void setLabel(QString label, int a = AlignLeft | AlignTop);
void setRange(double lower, double upper, double step = 1, bool slider = true);
void setSpecialValueText(const QString &text);
double value() const;
```

처리부

```
void setPrefix(QString prefix);
void setSuffix(QString suffix);
void setValue(double);
```

신호

void valueChanged(double);

그림 19-14에 보여준것처럼 입력창문은 값을 선택하는데 쓰이는 미끄럼띠를 포함한다. 결음값을 지정하는데 미끄럼띠나 스핀단추를 리용할수 있다.

```
/* showdoublenuminput.cpp */
#include <kapplication.h>
#include <knuminput.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showdoublenuminput");
    KDoubleNumInput *doublenuminput = new KDoubleNumInput(980);
    doublenuminput->setRange(100.0,1000.0,10,true);
    doublenuminput->show();
    app.setMainWidget(doublenuminput);
    return(app.exec());
}
```

그림 19-14. 미끄럼띠를 가지는 KDoubleNumInput창문부품

KDualColorButton

KDualColorButton창문부품은 매개가 각이한 색을 가지는 한쌍의 겹침단추들을 현시하며 사용자가 단추를 선택할 때 색선택대화칸이 열리게 함으로써 마우스에 응답한다.

머리부파일

```
#include <kdualcolorbtn.h>
```

기초클라스

QObject QPaintDevice QWidget Qt

구성자

```
KDualColorButton(QWidget *parent = 0, const char *name = 0);
KDualColorButton(const QColor &fgColor, const QColor &bgColor, QWidget *parent = 0,
        const char *name = 0);
```

메쏘드

```
QColor background();
DualColor current();
```

QColor currentColor();

QColor foreground();

virtual QSize sizeHint() const;

처리부

void slotSetBackground(const QColor &c);

void slotSetCurrent(KDualColorButton::DualColor s);

void slotSetCurrentColor(const QColor &c);

void slotSetForeground(const QColor &c);

신호

void bgChanged(const QColor &c);

void currentChanged(KDualColorButton::DualColor s);

void fgChanged(const QColor &c);

렬거형

enum DualColor { Foreground, Background };

다음의 실례는 그림 19-15에서 그려진 2개의 색단추를 현시한다. 단추들중 하나를 두번 찰칵하면 사용자가 색을 변경할수 있는 색대화칸창문을 펼친다.

```
/* showdualcolorbutton.cpp */
#include <kapplication.h>
#include <kdualcolorbtn.h>
int main(int argc,char **argv)
{
   KApplication app(argc,argv, "showdualcolorbutton");
   KDualColorButton *dualcolorbutton = new KDualColorButton(QColor("red"),QColor("green"));
   dualcolorbutton->show();
   app.setMainWidget(dualcolorbutton);
   return(app.exec());
}
```

그림 19-15. 2가지 색중 하나의 선택을 허용하는 KDualColorButton창문부품

KEdFind

KEdFind는 2개의 파라메터에 의해 탐색하려는 문자렬을 받아들이는 대화칸이다.

머리부파일

```
#include <keditcl.h>
```

기초클라스

KDialog KDialogBase QDialog QObject QPaintDevice QWidget Qt

구성자

```
KEdFind(QWidget *parent = 0, const char *name = 0, bool modal = true);
```

메쏘드

```
bool case_sensitive();
QString getText();
bool get_direction();
void setText(QString string);
```

신호

```
void done();
void search();
```

다음의 실례는 그림 19-16에 보여주는 KEdFind대화칸을 현시한다.

```
/* showedfind.cpp */
#include <kapplication.h>
#include <keditcl.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showedfind");
    KEdFind *edfind = new KEdFind();
    edfind->setText("The search string");
    edfind->show();
    return(app.exec());
}
```
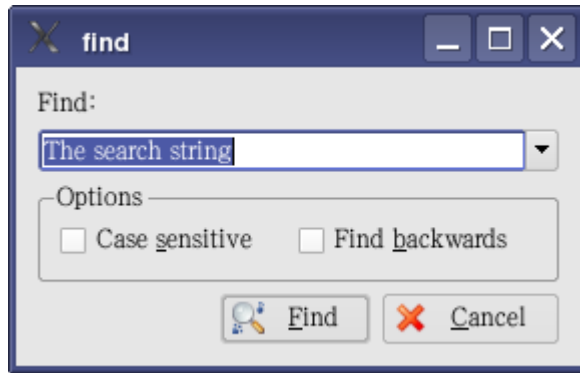
그림 19-16. 문자렬탐색을 하는데 필요한 정보를 요구하는 KEdFind대화칸

KEdGotoLine

KEdGotoLine창문부품은 행번호에 의해 본문을 탐색할수 있게 한다.

머리부파일

```
#include <keditcl.h>
```

기초클라스

KDialog KDialogBase QDialog QObject QPaintDevice QWidget Qt

구성자

```
KEdGotoLine(QWidget *parent = 0, const char *name = 0, bool modal = true);
```

메쏘드

```
int getLineNumber();
```

처리부

```
void selected(int);
```

다음의 실례는 그림 19-17에 보여주는 KEdGotoLine대화칸을 현시한다.

```
/* showedgotoline.cpp */
#include <kapplication.h>
#include <keditcl.h>
int main(int argc,char **argv)
{
   KApplication app(argc,argv, "showedgotoline");
   KEdGotoLine *edgotoline = new KEdGotoLine();
   edgotoline->show();
   return(app.exec());
}
```
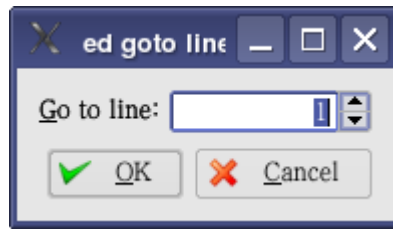
그림 19-17. 탐색본문의 행번호를 요구하는 KEdGotoLine대화칸

KEdit

KEdit창문부품은 간단한 문서편집기이다.

머리부파일

```
#include <keditcl.h>
```

기초클라스

QFrame QMultiLineEdit QObject QPaintDevice QTableView QWidget Qt

구성자

```
KEdit(QWidget *_parent = NULL, const char *name = NULL);
```

메쏘드

```
void cleanWhiteSpace();
int currentColumn();
int currentLine();
void doGotoLine();
void insertText(QTextStream *);
void installRBPopup(QPopupMenu *);
bool isModified();
QString markedText();
bool repeatSearch();
void replace();
void saveText(QTextStream *);
void search();
void selectFont();
void setModified(bool = true);
void spellcheck_start();
void spellcheck_stop();
```

처리부

```
void computePosition();
void corrected(QString originalword, QString newword, unsigned pos);
```

```
void misspelling(QString word, QStringList *, unsigned pos);
void repaintAll();
void replace_all_slot();
void replace_search_slot();
void replace_slot();
void replacedone_slot();
void search_slot();
void searchdone_slot();
```

신호

```
void CursorPositionChanged();
void gotUrlDrop(QDropEvent *e);
void toggle_overwrite_signal();
```

렬거형

```
enum (anon) { NONE, FORWARD, BACKWARD };
```

다음의 실례는 그림 19-18에 보여주는 편집창문을 현시한다.

```
/* showedit.cpp */
#include <kapplication.h>
#include <keditcl.h>
int main(int argc,char **argv)
{
   KApplication app(argc,argv, "showedit");
   KEdit *edit = new KEdit();
   edit->show();
   app.setMainWidget(edit);
   return(app.exec());
}
```
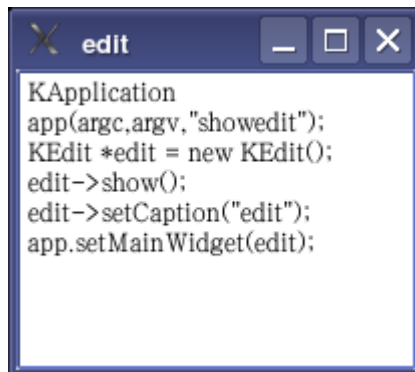
그림 19-18. 기본문서편집기인 KEdit창문부품

KEdReplace

KEdReplace창문부품은 문서편집기의 탐색 및 교체대화칸이다.

머리부파일

```
#include <keditcl.h>
```

기초클라스

KDialog KDialogBase QDialog QObject QPaintDevice QWidget Qt

구성자

```
KEdReplace(QWidget *parent = 0, const char *name = 0, bool modal = true);
```

메쏘드

```
bool case_sensitive();
QString getReplaceText();
QString getText();
bool get_direction();
void setText(QString);
```

신호

```
void done();
void find();
void replace();
void replaceAll();
```

다음의 실례는 그림 19-19에 보여주는 KEdReplace대화칸을 현시한다.

```
/* showedreplace.cpp */
#include <kapplication.h>
#include <keditcl.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showedreplace");
    KEdReplace *edreplace = new KEdReplace();
    edreplace->show();
    return(app.exec());
}
```
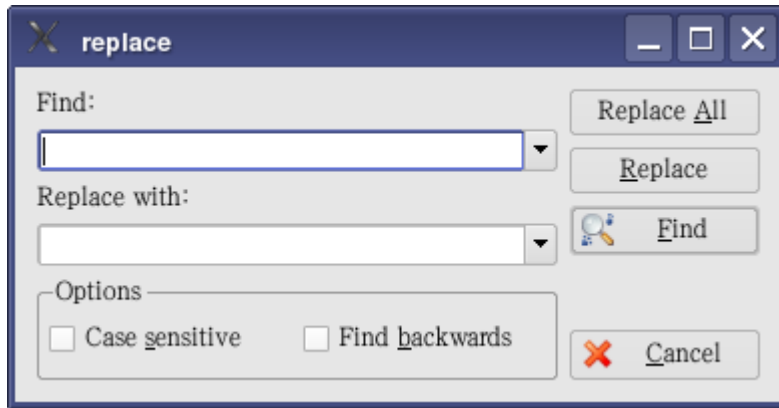
그림 19-19. 문자렬의 탐색 및 교체를 위한 KEdReplace대화칸

KFileDialog

KFileDialog창문부품은 파일들을 열람하고 선택하는데 사용되는 대화칸이다.

머리부파일

```
#include <kfiledialog.h>
```

기초클라스

KDialog KDialogBase QDialog QObject QPaintDevice QWidget Qt

구성자

```
KFileDialog(const QString &urlName, const QString &filter, QWidget *parent, const char *name,
    bool modal);
```

메쏘드

```
KURL baseURL() const;
QString currentFilter() const;
static QString getExistingDirectory(const QString &url = QString::null, QWidget *parent = 0,
    const QString &caption = QString::null);
static QString getOpenFileName(const QString &dir = QString::null,
    const QString &filter = QString::null, QWidget *parent = 0,
    const QString &caption = QString::null);
static QStringList getOpenFileNames(const QString &dir = QString::null,
    const QString &filter = QString::null, QWidget *parent = 0,
    const QString &caption = QString::null);
static KURL getOpenURL(const QString &url = QString::null, const QString &filter = QString::null,
    QWidget *parent = 0, const QString &caption = QString::null);
List getOpenURLs(const QString &url = QString::null, const QString &filter = QString::null,
    QWidget *parent = 0, const QString &caption = QString::null);
```

```
        static QString getSaveFileName(const QString &dir = QString::null,
              const QString &filter = QString::null, QWidget *parent = 0,
              const QString &caption = QString::null);
        static KURL getSaveURL(const QString &url = QString::null, const QString &filter = QString::null,
              QWidget *parent = 0, const QString &caption = QString::null);
        Mode mode() const;
        QString selectedFile() const;
        QStringList selectedFiles() const;
        KURL selectedURL() const;
        List selectedURLs() const;
        void setFilter(const QString &filter);
        void setLocationLabel(const QString &text);
        void setMode(KFile::Mode m);
        void setPreviewWidget(const QWidget *w);
        void setSelection(const QString &name);
        void setURL(const KURL &url, bool clearforward = true);
        virtual void show();
        KToolBar *toolBar() const;
```

신호

```
        void fileHighlighted(const QString &);
        void fileSelected(const QString &);
        void filterChanged(const QString &filter);
        void historyUpdate(bool, bool);
```

다음의 실례는 그림 19-20에 보여주는 파일대화칸창문을 현시한다. 구성자의 첫 인수는 파일탐색을 시작하는 등록부의 이름이다. 이 실례와 같이 NULL등록부가 지정되면 파일이 선택될 때까지 현재작업등록부가 사용되고 그다음부터 파일이 선택된 마지막 등록부가 기동시에 사용된다. 둘째 인수는 어느 파일들을 목록에 포함해야 하는가를 결정하는 려과기이다. 등록부들은 항상 포함되지만 파일들은 그 이름이 현재 선택된 정규식려과기와 일치하면 포함된다.

```
/* showfiledialog.cpp */
#include <kapplication.h>
#include <kfiledialog.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showfiledialog");
    QString filter("*.cpp|C++ Source Files\n"
```

```
        "*.h|C and C++ Header Files\n"
        "*.awk\n"
        "*.o|Object files\n"
        "*|All Files");
    KFileDialog *filedialog = new KFileDialog(0,filter, 0, "filedialog",FALSE);
    filedialog->show();
    return(app.exec());
}
```
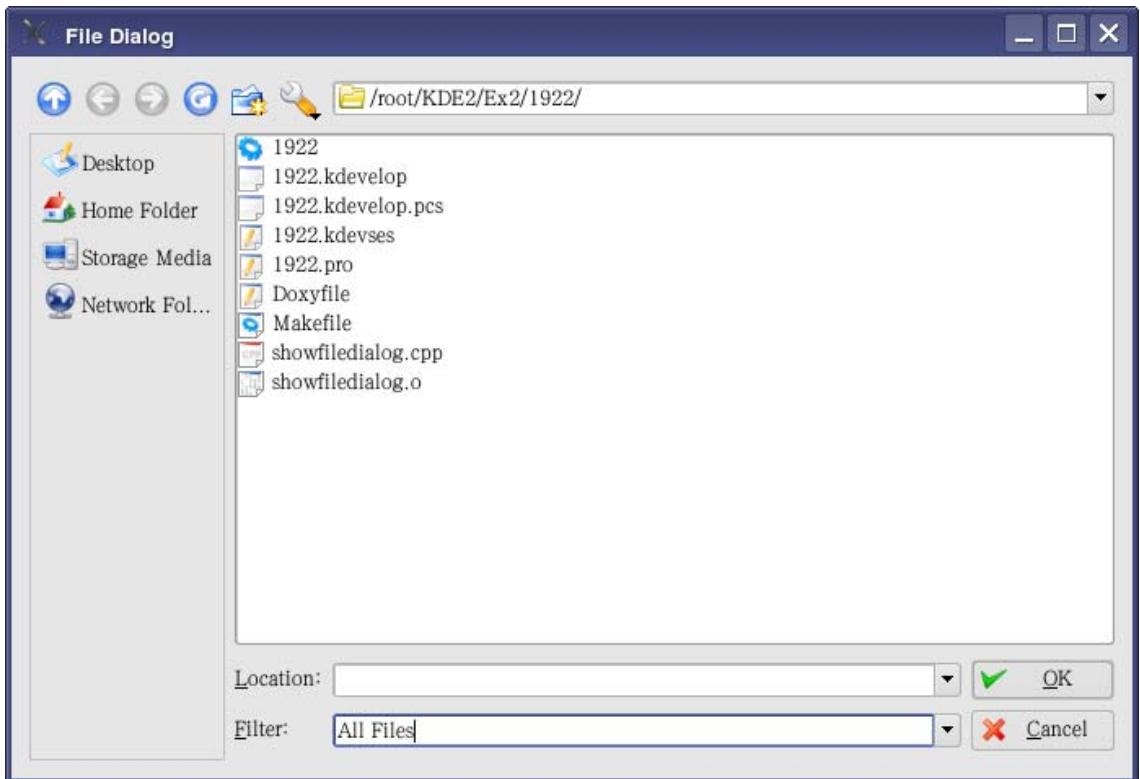
그림 19-20. 파일과 사용가능한 려과기들의 목록을 현시하는 KFileDialog창문부품

KFontChooser

KFontChooser는 사용자가 서체를 대화칸적으로 선택할수 있는 창문부품이다.

머리부파일

```
#include <kfontdialog.h>
```

기초클라스

QObject QPaintDevice QWidget Qt

구성자

```
KFontChooser(QWidget *parent = 0L, const char *name = 0L, bool onlyFixed = false,
```

const QStringList &fontList = QStringList () , bool makeFrame = true , int visibleListSize = 8);

메쏘드

void enableColumn(int column, bool state);

QFont font();

static void getFontList(QStringList &list, const char *pattern);

static QString getXLFD(const QFont &theFont);

QString sampleText();

void setFont(const QFont &font, bool onlyFixed = false);

void setSampleText(const QString &text);

virtual QSize sizeHint(void) const;

신호

void fontSelected(const QFont &font);

렬거형

enum FontColumn { FamilyList=0x01, StyleList=0x02, SizeList=0x04 };

5장에 KFontChooser를 사용하는 실례가 있다.

KFontDialog

KFontDialog는 사용자가 서체를 선택할수 있는 대화칸이다.

머리부파일

#include <kfontdialog.h>

기초클라스

KDialog KDialogBase QDialog QObject QPaintDevice QWidget Qt

구성자

KFontDialog(QWidget *parent = 0L, const char *name = 0, bool modal = false, bool onlyFixed = fa
lse,

const QStringList &fontlist = QStringList () , bool makeFrame = true);

메쏘드

QFont font();

static int getFont(QFont &theFont, bool onlyFixed = false, QWidget *parent = 0L,

bool makeFrame = true);

static int getFontAndText(QFont &theFont, QString &theString, bool onlyFixed = false,

QWidget *parent = 0L, bool makeFrame = true);

void setFont(const QFont &font, bool onlyFixed = false);

신호

void fontSelected(const QFont &font);

KFontDialog을 사용하는 실례들은 10장에 있다.

KFormulaEdit

이 KFormulaEdit창문부품은 공식을 현시하고 편집하는데 사용된다.

머리부파일

```
#include <kformulaedit.h>
```

기초클라스

QObject QPaintDevice QWidget Qt

구성자

```
KFormulaEdit(QWidget *parent = 0, const char *name = 0, WFlags f = 0, bool restricted = false);
```

메쏘드

```
void enableSizeHintSignal(bool b);
KFormula *getFormula() const;
void redraw(int all = 1);
void setExtraChars(QString c);
void setText(QString text);
void setUglyForm(QString ugly);
virtual QSize sizeHint() const;
virtual QSizePolicy sizePolicy() const;
QString text() const;
QString uglyForm() const;
```

처리부

```
void insertChar(int c);
```

신호

```
void formulaChanged(const QString &);
void sizeHint(QSize);
```

다음의 실례는 그림 19-21에 보여주는 공식을 현시하는데 KFormulaEdit창문부품을 사용한다. 공식은 KFormula의 fromUgly메쏘드에 의해 C형표현(“ugly”형식)으로부터 형식화된다. 공식은 구성자에서 restricted기발을 TRUE로 설정하면 평가가능한 식을 편집할수 있다.

```
/* showformulaedit.cpp */
#include <kapplication.h>
#include <kformulaedit.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showformulaedit");
```

```
    KFormulaEdit *formulaedit = new KFormulaEdit(0,0,0,TRUE);
    QString fn = KFormula::fromUgly("44.2 - k*(99.2 + 544) ");
    formulaedit->setText(fn);
    formulaedit->show();
    app.setMainWidget(formulaedit);
    return(app.exec());
}
```
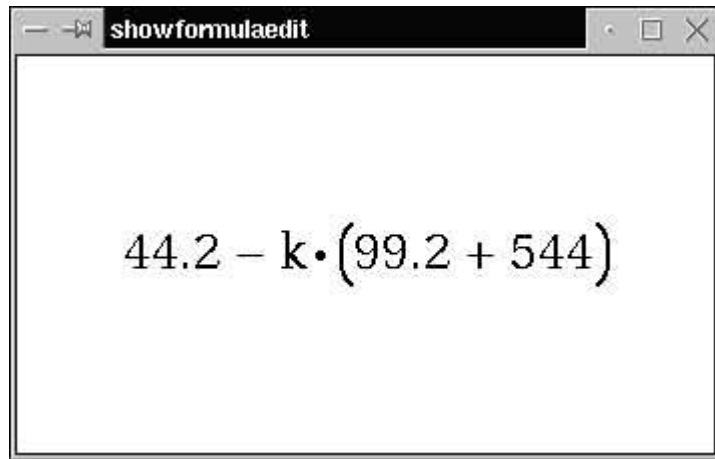
그림 19-21. 공식을 현시하는 KFormulaEdit창문부품

KFormulaToolBar

KFormulaToolBar는 KFormulaEdit창문부품에 특별한 형식화와 문자입력을 제공하는 도구띠창문부품이다.

머리부파일

#include <kformulatoolbar.h>

기초클라스

KToolBar QFrame QObject QPaintDevice QWidget Qt

구성자

KFormulaToolBar(QWidget *parent = 0L, const char *name = 0L, int _item_size = - 1);

메쏘드

void connectToFormula(KFormulaEdit *formula);

다음의 실레는 그림 19-22에 보여주는 KFormulaToolBar를 현시한다.

```
/* showformulatoolbar.cpp */
#include <kapplication.h>
#include <kformulatoolbar.h>
int main(int argc,char **argv)
```

```
{
    KApplication app(argc,argv, "showformulatoolbar");
    KFormulaToolBar *formulatoolbar = new KFormulaToolBar();
    formulatoolbar->show();
    app.setMainWidget(formulatoolbar);
    return(app.exec());
}
```

그림 19-22 .KFormulaToolbar창문부품

KGradientSelector

KGradientSelector창문부품은 2가지 색사이의 경사도를 현시한다. 그것은 마우스를 극값들사이의 색을 선택하는데 사용할수 있다.

머리부파일

```
#include <kselect.h>
```

기초클라스

KSelector QObject QPaintDevice QRangeControl QWidget Qt

구성자

```
KGradientSelector(Orientation o, QWidget *parent = 0L, const char *name = 0L);
```

메쏘드

```
void setColors(const QColor &col1, const QColor &col2);
void setText(const QString &t1, const QString &t2);
```

다음의 실례는 그림 19-23에 보여주는것처럼 붉은색과 흰색사이의 경사색들을 보여준다. 2개 극값들사이의 대조를 보여주는 두개의 본문을 보여준다. 그림의 바닥에 현재선택점을 표식하는 QRangeControl로부터 계승된 지시자가 있다.

```
/* showgradientselector.cpp */
#include <kapplication.h>
#include <kselect.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showgradientselector");
    KGradientSelector *gradientselector = new KGradientSelector(KSelector::Horizontal);
    gradientselector->setColors(QColor("red"),QColor("white"));
```

```
    gradientselector->setText("White on Red","Red on White");
    gradientselector->show();
    app.setMainWidget(gradientselector);
    return(app.exec());
}
```
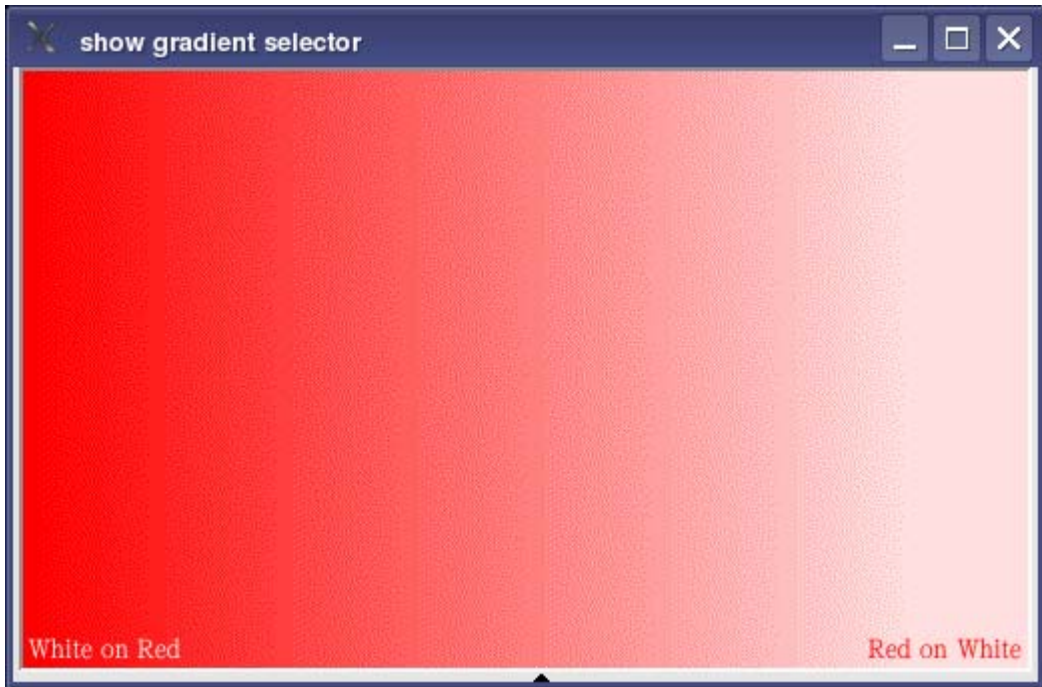
그림 19-23. 색경사도와 본문을 둘다 표시하는 KGradientSelector창문부품

KHSSelector

KHSSelector창문부품은 사용자가 마우스로 색 또는 색의 농도를 선택할수 있게 한다.

머리부파일

```
#include <kcolordlg.h>
```

기초클라스

KXYSelector QObject QPaintDevice QWidget Qt

구성자

```
KHSSelector(QWidget *parent);
```

다음의 실례는 그림 19-24에 보여주는 다색창문을 현시한다. 색선택은 마우스를 가지고 KXYSelector로부터 계승된 지시자를 적당한 장소에 놓음으로써 수행된다.

```
/* showhsselector.cpp */
#include <kapplication.h>
#include <kcolordlg.h>
```

```
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showhsselector");
    KHSSelector *hsselector = new KHSSelector(0);
    hsselector->show();
    app.setMainWidget(hsselector);
    return(app.exec());
}
```
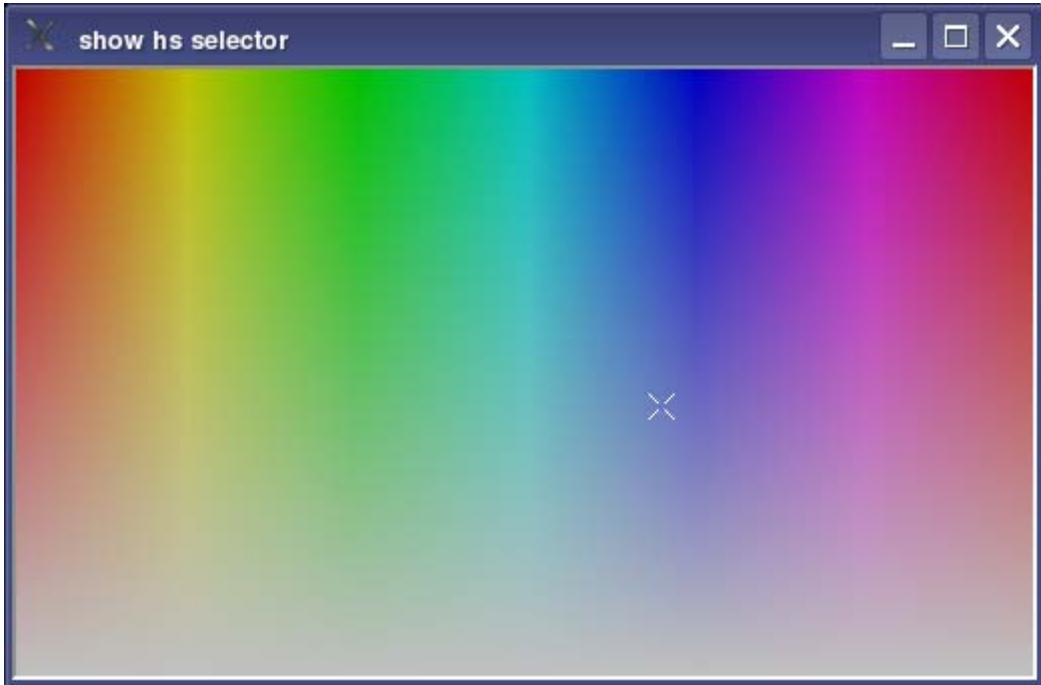
그림 19-24. 현재 선택한 색점을 표시하는 KHSSelector창문부품

KHTMLView

KHTMLView창문부품은 웨브페지를 현시하는데 쓰인다.

머리부파일

```
#include <khtmlview.h>
```

기초클라스

QFrame QObject QPaintDevice QScrollView QWidget Qt

구성자

```
KHTMLView(KHTMLPart *part, QWidget *parent, const char *name = 0);
```

메쏘드

```
bool dndEnabled() const;
```

```
int frameWidth() const;
bool gotoNextLink();
bool gotoPrevLink();
bool hasSelection() const;
void layout(bool force = false);
int marginHeight();
int marginWidth() const;
KHTMLPart *part() const;
void print();
QString selectedText() const;
void setDNDEnabled(bool b);
void setMarginHeight(int y);
void setMarginWidth(int x);
void setURLCursor(const QCursor &c);
void toggleActLink(bool);
const QCursor & urlCursor() const;
static const QList<KHTMLView> *viewList();
```

신호

```
void selectionChanged();
```

다음의 실례는 그림 19-25에 보여주는 웨브페지를 현시하는데 KHTMLView객체를 사용한다. 그것은 URL을 보관하는 KURL객체를 만들어서 수행한다. 이 실례에서 URL은 국부디스크에 있으나 인터네트상의 임의의곳일수 있다. KURL객체는 실제 웨브페지를 열 때 KHTMLPart객체에 의해서 사용된다. 독자적인 구성자를 사용하지 않고 이미 페지를 현시하도록 설정된 KHTMLView객체를 돌려주려고 할 때 KHTMLPart객체의 view()메쏘드가 호출된다. KHTMLView의 크기가 설정되고 창문부품이 현시될 때 페지 그 자체가 현시된다.

```
/* showhtmlview.cpp */
#include <kapplication.h>
#include <iostream.h>
#include <kurl.h>
#include <khtml_part.h>
#include <khtmlview.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showhtmlview");
    KURL kurl = "file:/home/testpage.html";
    KHTMLPart *part = new KHTMLPart();
```

```
    if(!part->openURL(kurl))
        cout << "The URL failed to open" << endl;
    KHTMLView *htmlview = part->view();
    htmlview->resize(400,200);
    htmlview->show();
    app.setMainWidget(htmlview);
    return(app.exec());
}
```
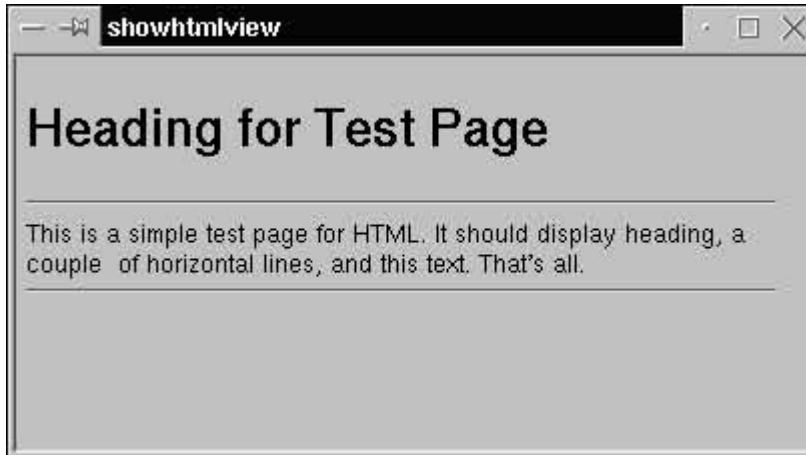
그림 19-25. KHTMLPart객체로부터 웨브페지를 현시하는 KHTMLVIew창문부품

KIconButton

KIconButton은 KIconDialog창문을 펼친 다음 선택된 그림기호를 현시하는 단추창문부품이다.

머리부파일

```
#include <kicondialog.h>
```

기초클라스

QButton QObject QPaintDevice QPushButton QWidget Qt

구성자

```
KIconButton(QWidget *parent = 0L, const char *name = 0L);
KIconButton(KIconLoader *loader, QWidget *parent, const char *name = 0L);
```

메쏘드

```
const QString icon();
void setIcon(QString icon);
void setIconType(int group, int context, bool user = false);
```

신호

void iconChanged(QString icon);

다음 실례는 그림기호를 포함한 KIconButton을 현시한다. 이 실례는 그림 19-26과 같이 setIcon()을 호출함으로써 시작그림기호를 지정한다. 그림기호가 지정되지 않았으면 단추는 처음에 비여있고 KIconDialog로부터 그림기호가 선택될 때까지 그대로 남아있다.

```
/* showiconbutton.cpp */
#include <kapplication.h>
#include <kicondialog.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showiconbutton");
    KIconButton *iconbutton = new KIconButton();
    iconbutton->setIcon("go");
    iconbutton->show();
    app.setMainWidget(iconbutton);
    return(app.exec());
}
```

그림 19-26. 현재 선택된 그림기호를 현시하는 KIconButton

KIconDialog

KIconDialog창문부품은 그림기호를 선택하는데 쓰이는 대화칸이다.

머리부파일

#include <kicondialog.h>

기초클라스

KDialog KDialogBase QDialog QObject QPaintDevice QWidget Qt

구성자

KIconDialog(QWidget *parent = 0L, const char *name = 0L);

KIconDialog(KIconLoader *loader, QWidget *parent = 0, const char *name = 0);

메쏘드

QString selectIcon(int group = KIcon::Desktop, int context = KIcon::Application, bool user = false);

다음 실례는 KIconDialog창문을 현시한다. 그림 19-27은 체계응용프로그람의 그림기호들을 현시하고있다.

```
/* showicondialog.cpp */
#include <kapplication.h>
#include <kicondialog.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showicondialog");
    KIconDialog *icondialog = new KIconDialog();
    icondialog->show();
    return(app.exec());
}
```
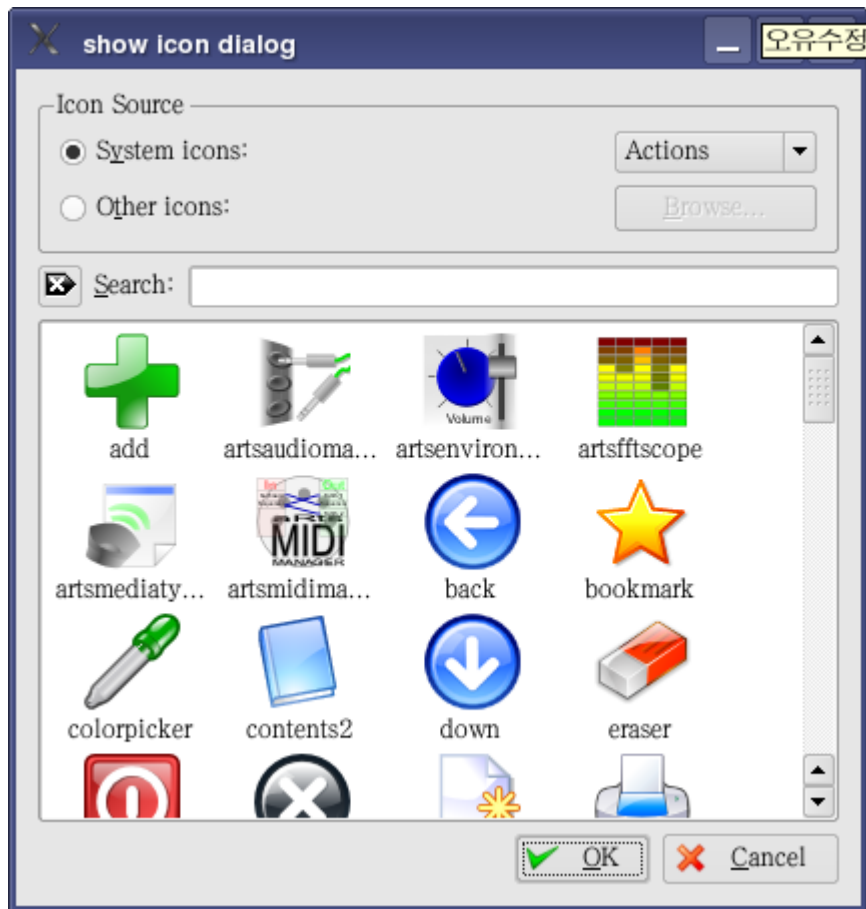
그림 19-27. 체계응용프로그람 그림기호들을 현시하는 KIconDialog

KIconView

KIconView창문부품은 사용자가 선택할수 있는 그림기호들의 모임을 현시한다. 이 창문부품은 마우스단추조종과 선택에 표준KDE환경을 사용하도록 QIconView를 확장한다.

머리부파일

```
#include <kiconview.h>
```

기초클라스

QFrame QIconView QObject QPaintDevice QScrollView QWidget Qt

파생클라스

KFileIconView KIconCanvas

구성자

```
KIconView(QWidget *parent = 0, const char *name = 0, WFlags f = 0);
```

신호

```
void doubleClicked(QIconViewItem *item, const QPoint &pos);
void executed(QIconViewItem *item);
void executed(QIconViewItem *item, const QPoint &pos);
```

다음 실례는 그림 19-28에 보여주는 5개 그림기호를 현시한다. 첫째 그림기호는 픽스매프도 본문도 가지지 않으므로 기정픽스매프를 사용하여 표식을 얻는다. 다음 두 그림기호도 역시 기정픽스매프를 사용하지만 둘다 표식에 본문을 가지고있다. 마지막 2개 그림기호는 픽스매프와 표식을 모두 가지며 “Flag”로 표시된 그림기호가 마우스에 의해 선택되였다.

```
/* showiconview.cpp */
#include <kapplication.h>
#include <kiconview.h>
int main(int argc,char **argv)
{
    QIconViewItem *item;
    KApplication app(argc,argv,”showiconview”);
    KIconView *iconview = new KIconView();
    item = new QIconViewItem(iconview);
    item = new QIconViewItem(iconview, "Icon Label");
    item = new QIconViewItem(iconview, "Icon With\nLong Label");
    QPixmap flag("flag.png");
    item = new QIconViewItem(iconview, "Flag",flag);
    QPixmap idea("idea.png");
    item = new QIconViewItem(iconview, "Idea",idea);
    iconview->show();
    app.setMainWidget(iconview);
    return(app.exec());
}
```
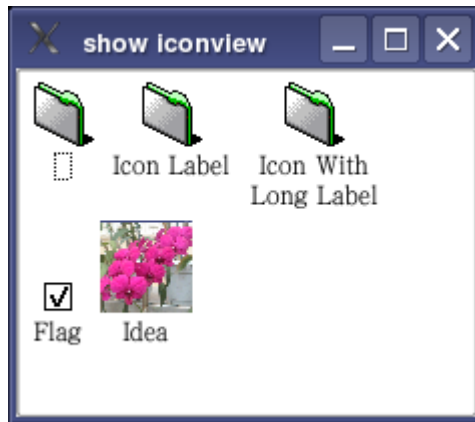
그림 19-28. 5개 그림기호를 현시하는 KIconView창문부품

KImageTrackLabel

KImageTrackLabel창문부품은 QLabel을 확장하여 마우스동작을 추적하고 조종능력을 추가한다.

머리부파일

#include <kaboutdialog.h>

기초클라스

QFrame QLabel QObject QPaintDevice QWidget Qt

구성자

KImageTrackLabel(QWidget *parent, const char *name = 0, WFlags f = 0);

신호

void mouseTrack(int mode, const QMouseEvent *e);

렬거형

enum MouseMode { MousePress=1, MouseRelease, MouseDoubleClick, MouseMove };

다음 실례는 그림 19-29에 보여주는 간단한 본문창문부품으로서 KImageTrackLabel창문부품을 현시한다. 처리부가 mouseTrack()에 련결되였다면 모든 마우스작용이 통보된다.

```
/* showimagetracklabel.cpp */
#include <kapplication.h>
#include <kaboutdialog.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showimagetracklabel");
    KImageTrackLabel *imagetracklabel = new KImageTrackLabel(0);
    imagetracklabel->setText("Mouse Tracking Label");
    imagetracklabel->show();
```

```
    app.setMainWidget(imagetracklabel);
    return(app.exec());
}
```

그림 19-29. 마우스를 추적하는 표식자로써 KImageTrackLabel

KIntNumInput

KIntNumInput창문부품은 사용자의 옹근수값입력을 받아들이고 확인하기 위한것이다.

머리부파일

```
#include <knuminput.h>
```

기초클라스

KNumInput QObject QPaintDevice QWidget Qt

구성자

```
KIntNumInput(int value, QWidget *parent = 0, int base = 10, const char *name = 0);
KIntNumInput(KNumInput *below, int value, QWidget *parent = 0, int base = 10,
        const char *name = 0);
```

메쏘드

```
virtual QSize minimumSizeHint() const;
virtual void setLabel(QString label, int a = AlignLeft | AlignTop);
void setRange(int lower, int upper, int step = 1, bool slider = true);
void setSpecialValueText(const QString &text);
int value() const;
```

처리부

```
void setEditFocus(bool mark = true);
void setPrefix(QString prefix);
void setSuffix(QString suffix);
void setValue(int);
```

신호

```
void valueChanged(int);
```

그림 19-30과 같이 입력창문은 미끄럼띠 또는 스핀단추에 의하여 기동할수 있고 둘다 값을 선택하는데 사용된다. 또한 값은 자료창문에 직접 입력된다.

```
/* showintnuminput.cpp */
#include <kapplication.h>
```

```
#include <knuminput.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showintnuminput");
    KIntNumInput *intnuminput = new KIntNumInput(980);
    intnuminput->setRange(100,1000,10,true);
    intnuminput->show();
    app.setMainWidget(intnuminput);
    return(app.exec());
}
```

그림 19-30. 스핀단추와 미끄럼띠를 모두 가진 KIntNumInput창문부품

KIntSpinBox

KIntSpinBox창문부품은 사용자의 옹근수값입력을 받아들이고 확인하기 위한것이다.

머리부파일

```
#include <knuminput.h>
```

기초클라스

QFrame QObject QPaintDevice QRangeControl QSpinBox QWidget Qt

구성자

```
KIntSpinBox(int lower, int upper, int step, int value, int base = 10, QWidget *parent = 0,
        const char *name = 0);
```

메쏘드

```
void setEditFocus(bool mark);
```

다음 실례는 그림 19-31에 보여주는 KIntSpinBox를 만든다. 값은 스핀칸을 사용하여 걸음값씩 수를 변경하거나 자료창문에 직접 값을 입력하여 설정한다.

```
/* showintspinbox.cpp */
#include <kapplication.h>
#include <knuminput.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showintspinbox");
    KIntSpinBox *intspinbox = new KIntSpinBox(100,1000,10,980);
```

```
    intspinbox->show();
    app.setMainWidget(intspinbox);
    return(app.exec());
}
```
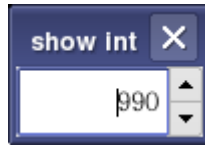
그림 19-31. 스핀단추를 가지는 KIntSpinBox창문부품

KKeyButton

KKeyButton창문부품은 건반단추와 같이 보이는 QButton이다.

머리부파일

#include <kkeydialog.h>

기초클라스

QButton QObject QPaintDevice QPushButton QWidget Qt

구성자

KKeyButton(const char *name = 0, QWidget *parent = 0);

메쏘드

void setEdit(bool edit);

void setText(const QString &text);

다음 실례프로그람은 제일 웃준위창문부품으로서 KKeyButton창문부품을 사용하며 그림 19-32에서 보여준다.

```
/* showkeybutton.cpp */
#include <kapplication.h>
#include <kkeydialog.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showkeybutton");
    KKeyButton *keybutton = new KKeyButton();
    keybutton->setText("Key Button");
    keybutton->show();
    app.setMainWidget(keybutton);
    return(app.exec());
}
```

그림 19-32. 건반단추모양의 KKeyButton창문부품

KLed

KLed창문부품은 각이한 모양과 색의 지시자로 쓰이는 LED를 현시한다.

머리부파일

```
#include <kled.h>
```

기초클라스

QObject QPaintDevice QWidget Qt

구성자

```
KLed(const QColor &col = Qt::green, QWidget *parent = 0, const char *name = 0);
KLed(const QColor &col, KLed::State st, KLed::Look look, KLed::Shape shape, QWidget *parent = 0,
    const char *name = 0);
```

메쏘드

```
const QColor color() const;
int getDarkFactor() const;
Look look() const;
void setColor(const QColor &color);
void setDarkFactor(int darkfactor);
void setLook(Look look);
void setShape(Shape s);
void setState(State state);
State state() const;
void toggleState();
```

처리부

```
void off();
void on();
void toggle();
```

렬거형

```
enum State { Off, On, NoOfStates };
enum Shape { NoShape, Rectangular, Circular, NoOfShapes=Circular };
enum Look { NoLook, Flat, Raised, Sunken, NoOfLooks=Sunken };
```

KLed실례를 그림 19-33에 보여준다. 그것은 도드라져나온 원형지시자인데 기정색 즉 록

색으로 색칠된다.

```
/* showled.cpp */
#include <kapplication.h>
#include <kled.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showled");
    KLed *led = new KLed();
    led->setLook(KLed::Raised);
    led->setShape(KLed::Circular);
    led->setState(KLed::On);
    led->resize(10,10);
    led->show();
    app.setMainWidget(led);
    return(app.exec());
}
```
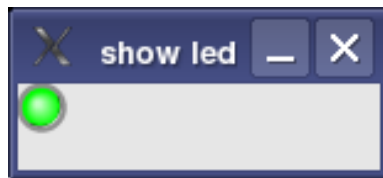
그림 19-33. 도드라져나온 둥근 KLed창문부품

KLineEdit

KLineEdit창문부품은 단일행본문편집기이다.

머리부파일

```
#include <klineedit.h>
```

기초클라스

KCompletionBase QLineEdit QObject QPaintDevice QWidget Qt

파생클라스

KRestrictedLine

구성자

```
KLineEdit(const QString &string, QWidget *parent, const char *name = 0);
KLineEdit(QWidget *parent = 0, const char *name = 0);
```

메쏘드

```
void cursorAtEnd();
```

```
bool isContextMenuEnabled() const;
virtual void setCompletionMode(KGlobalSettings::Completion mode);
virtual void setEnableContextMenu(bool showMenu);
```

처리부

```
void rotateText(KeyBindingType);
```

신호

```
void completion(const QString &);
void nextMatch(KeyBindingType);
void previousMatch(KeyBindingType);
void returnPressed(const QString &);
```

다음 실례는 그림 19-34에서 보여주는 KLineEdit창문을 창조하고 현시한다. KLineEdit창문부품은 QLineEdit창문부품에 기초한다. KLineEdit에 추가된 특성으로서 선택적인 본문보완과 환경구성가능한 건결합이 있다.

```
#include <kapplication.h>
#include <klineedit.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showlineedit");
    KLineEdit *lineedit = new KLineEdit();
    lineedit->show();
    app.setMainWidget(lineedit);
    return(app.exec());
}
```
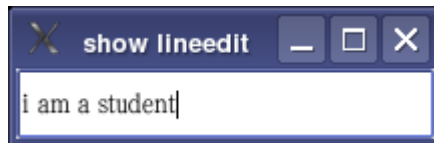
그림 19-34. QLineEdit을 계승한 KLineEdit창문부품

QLineEdit창문부품의 실례는 4장에 있다.

KLineEditDlg

KLineEditDlg는 단일행본문을 편집하기 위한 QLineEdit창문부품을 담고있는 대화칸이다.

머리부파일

```
#include <klineeditdlg.h>
```

기초클라스

```
KDialog KDialogBase QDialog QObject QPaintDevice QWidget Qt
```

구성자

KLineEditDlg(const QString &_text, const QString &_value, QWidget *parent);

메쏘드

static QString getText(const QString &_text, const QString &_value, bool *ok, QWidget *parent);

QString text();

처리부

void slotClear();

다음 실례는 그림 19-35와 같이 제목문자렬과 기정본문을 가진 KLineEditDlg창문을 현시한다.

```
/* showlineeditdlg.cpp */
#include <kapplication.h>
#include <klineeditdlg.h>
int main(int argc,char **argv)
{
    bool OK;
    QString str;
    KApplication app(argc,argv, "showlineeditdlg");
    str = KLineEditDlg::getText("The caption", "The editable text", &OK, 0);
    if(OK) {
        // The OK button was selected
    } else {
        // The CANCEL button was selected
    }
    return(app.exec());
}
```
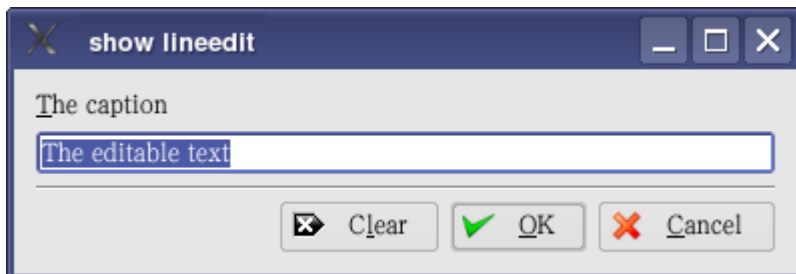
그림 19-35. QLineEdit창문부품을 가진 KLineEditDlg대화칸

KListBox

KListBox창문부품은 사용자가 마우스로 선택할수 있는 항목목록을 현시한다. 이 창문부품은 QListBox를 확장하여 마우스단추조종과 선택을 위한 표준KDE환경을 사용하게 한다.

머리부파일

```
#include <klistbox.h>
```

기초클라스

QFrame QListBox QObject QPaintDevice QScrollView QWidget Qt

파생클라스

KSplitList

구성자

```
KListBox(QWidget *parent = 0, const char *name = 0, WFlags f = 0);
```

신호

```
void doubleClicked(QListBoxItem *item, const QPoint &pos);
void executed(QListBoxItem *item);
void executed(QListBoxItem *item, const QPoint &pos);
```

다음의 실례는 그림 19-36에 보여주는 KListBox창문을 현시한다.

```
/* showlistbox.cpp */
#include <kapplication.h>
#include <klistbox.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showlistbox");
    KListBox *listbox = new KListBox();
    for(int i=0; i<20; i++) {
        QString str;
        str.sprintf("Selection %d\n",i);
        listbox->insertItem(str);
    }
    listbox->setMinimumWidth(120);
    listbox->show();
    app.setMainWidget(listbox);
    return(app.exec());
}
```
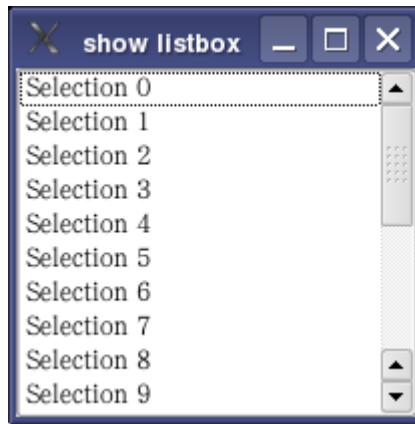
그림 19-36. 현재 선택한 항목을 현시하는 KListBox창문부품

KListView

KListView창문부품은 사용자가 마우스로 열람할수 있는 목록을 나무형식으로 현시한다. 이 창문부품은 QListView를 확장하여 마우스단추조작과 선택에 표준KDE환경을 사용하게 한다.

머리부파일

```
#include <klistview.h>
```

기초클라스

QFrame QListView QObject QPaintDevice QScrollView QWidget Qt

파생클라스

KApplicationTree KFileDetailView

구성자

```
KListView(QWidget *parent = 0, const char *name = 0);
```

메쏘드

```
virtual bool isExecuteArea(const QPoint &point);
```

신호

```
void doubleClicked(QListViewItem *item, const QPoint &pos, int c);
void executed(QListViewItem *item);
void executed(QListViewItem *item, const QPoint &pos, int c)
```

이 클라스는 QListView의 간단한 확장이며 거의 류사하게 동작한다.

18장에 QListView를 사용한 실례가 있다.

KMainWindow

KMainWindow창문부품은 표준KDE의 제일 웃준위창문이다.

머리부파일

#include <kmainwindow.h>

기초클라스

KXMLGUIBuilder KXMLGUIClient QObject QPaintDevice QWidget Qt

파생클라스

KDockMainWindow

구성자

KMainWindow(const char *name = 0L, WFlags f = WDestructiveClose);

메쏘드

int addToolBar(KToolBar *toolbar, int index = - 1);

static bool canBeRestored(int number);

static const QString classNameOfToplevel(int number);

virtual void createGUI(const QString &xmlfile = QString::null);

void enableStatusBar(KStatusBar::BarStatus stat = KStatusBar::Toggle);

void enableToolBar(KToolBar::BarStatus stat = KToolBar::Toggle, int id = 0);

virtual KXMLGUIFactory *guiFactory();

bool hasMenuBar();

bool hasStatusBar();

bool hasToolBar(int ID = 0);

QPopupMenu *helpMenu(const QString &aboutAppText = QString::null, bool showWhatsThis = true);

QWidget *indicator();

QRect mainViewGeometry() const;

KMenuBar *menuBar();

bool restore(int number);

void setEnableToolBar(KToolBar::BarStatus stat = KToolBar::Toggle, const QString &name = mainToo
lBar);

void setFrameBorderWidth(int);

void setIndicatorWidget(QWidget *indicator);

void setMaximumToolBarWraps(unsigned int wraps);

void setMenu(KMenuBar *menuBar);

void setStatusBar(KStatusBar *statusBar);

void setCentralWidget(QWidget *view, bool show_frame = TRUE);

virtual void show();

QSize sizeHint() const;

KStatusBar *statusBar();

KToolBar *toolBar(int ID = 0);

```
KToolBar *toolBar(const QString &name);
QWidget *view() const;
```

처리부

```
void appHelpActivated(void);
virtual void setCaption(const QString &caption);
virtual void setPlainCaption(const QString &caption);
```

6장에 KMainWindow창문부품의 실례가 있다.

KMenuBar

KMenuBar는 튀여나오기차림표들의 그룹사이의 관계를 관리할수 있는 수평띠이다. 이 창문부품은 QMenuBar를 확장하여 마우스단추조종과 선택에 표준KDE환경을 사용하게 한다.

머리부파일

```
#include <kmenubar.h>
```

기초클라스

QFrame QMenuBar QMenuData QObject QPaintDevice QWidget Qt

구성자

```
KMenuBar(QWidget *parent = 0, const char *name = 0);
```

다음 실례는 2장의 튀여나오기차림표를 포함하는 KMenuBar를 현시한다. 매개 튀여나오기차림표는 단일차림표항목을 담고 있다. 그림 19-37은 두번째 튀여나오기차림표를 가진 차림표띠를 보여준다.

```
/* showmenubar.cpp */
#include <kapplication.h>
#include <kmenubar.h>
#include <kpopupmenu.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showmenubar");
    KMenuBar *menubar = new KMenuBar();
    menubar->setSeparator(KMenuBar::InWindowsStyle);
    KPopupMenu* filePopup = new KPopupMenu();
    filePopup->insertItem("&Quit",&app,SLOT(quit()));
    menubar->insertItem("&File",filePopup);
    KPopupMenu* editPopup = new KPopupMenu();
    editPopup->insertItem("&Paste");
    menubar->insertItem("E&dit",editPopup);
```

```
    menubar->show();
    app.setMainWidget(menubar);
    return(app.exec());
}
```
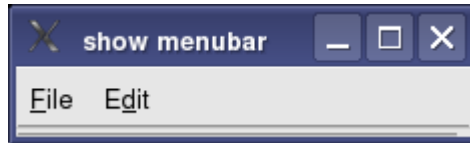
그림 19-37. 튀여나오기차림표를 보여주는 KMenuBar

KNumInput

KNumInput창문부품은 수입력창문부품을 실현하는데 쓰이는 기초클라스이다.

머리부파일

```
#include <knuminput.h>
```

기초클라스

QObject QPaintDevice QWidget Qt

파생클라스

KDoubleNumInput KIntNumInput

구성자

```
KNumInput(QWidget *parent = 0, const char *name = 0);
KNumInput(KNumInput *below, QWidget *parent = 0, const char *name = 0);
```

메쏘드

```
virtual void setLabel(QString label, int a = AlignLeft | AlignTop);
void setSteps(int minor, int major);
virtual QSize sizeHint() const;
QSizePolicy sizePolicy() const;
```

KPaletteTable

KPaletteTable창문부품은 사용자가 색을 선택할수 있게 한다.

머리부파일

```
#include <kcolordlg.h>
```

기초클라스

QObject QPaintDevice QWidget Qt

구성자

```
KPaletteTable(QWidget *parent, int minWidth = 210, int cols = 16);
```

메쏘드

```
void addToCustomColors(const QColor &);
void addToRecentColors(const QColor &);
QString palette();
```

처리부

```
void setPalette(const QString &paletteName);
```

신호

```
void colorSelected(const QColor &, const QString &);
```

다음 실례는 KPaletteTable창문부품을 현시한다. 그림 19-38과 같이 색을 선택하는 방법은 많다. 그림의 왼쪽에서 KPaletteTable창문부품은 직4각형의 색항목들을 현시하며 오른쪽에 있는 KPaletteTable창문부품은 이름에 의해 색목록을 현시한다.

```
/* showpalettetable.cpp */
#include <kapplication.h>
#include <kcolordlg.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showpalettetable");
    KPaletteTable *palettetable = new KPaletteTable(0);
    palettetable->show();
    app.setMainWidget(palettetable);
    return(app.exec());
}
```
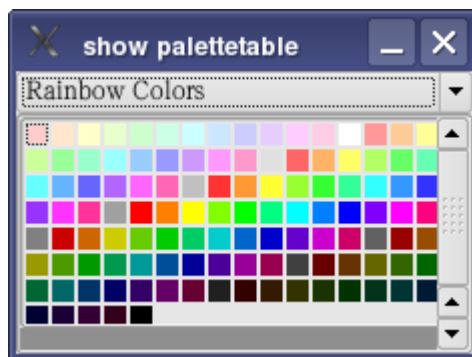
그림 19-38. 두가지 색선택방법을 보여주는 2개의 KPaletteTable창문부품

KPanelApplet

KPanelApplet는 KDE애플레트를 창조하는데 쓰이는 기초클라스이다.

머리부파일

```
#include <kpanelapplet.h>
```

기초클라스

```
QObject QPaintDevice QWidget Qt
```

구성자

```
KPanelApplet(QWidget *parent = 0, const char *name = 0);
```

메쏘드

```
virtual void about();
int actions();
bool flags();
virtual int heightForWidth(int width);
virtual void help();
void init(int &argc, char **argv);
Orientation orientation() const;
Position position() const;
virtual void preferences();
bool process(const QCString &fun, const QByteArray &data, QCString &replyType,
        QByteArray &replyData);
virtual void removedFromPanel();
void setActions(int a);
void setFlags(int f);
void updateLayout();
virtual int widthForHeight(int height);
```

렬거형

```
enum Actions { About=1, Help=2, Preferences=4 };
enum Flags { Stretch=1, TopLevel=2 };
enum Position { Left=0, Right, Top, Bottom };
```

15장에서 애플레트를 창조하는데 KPanelApplet클라스를 사용하는 실례가 있다.

KPasswordDialog

KPasswordDialog창문부품은 암호의 입력과 새로운 암호의 확인에 쓰인다.

머리부파일

```
#include <kpassdlg.h>
```

기초클라스

```
KDialog KDialogBase QDialog QObject QPaintDevice QWidget Qt
```

구성자

```
KPasswordDialog(int type, QString prompt, bool enableKeep = false, int extraBttn = 0);
```

메쏘드

```
void addLine(QString key, QString value);
static void disableCoreDumps();
static int getNewPassword(QCString &password, QString prompt);
static int getPassword(QCString &password, QString prompt, int *keep = 0L);
bool keep() const;
const char *password() const;
void setPrompt(QString prompt);
```

렬거형

```
enum Types { Password, NewPassword };
```

다음 실례는 사용자로부터 암호를 요구하는데 KPasswordDialog창문부품을 사용하는 방법을 보여준다. 그림 19-39와 같이 입력한 문자렬은 별표들의 렬로 나타난다. 메쏘드 getNewPassword()를 getPassword()대신에 사용한다면 암호는 두번 입력하여 확인해야 한다.

```
/* showpassworddialog.cpp */
#include <kapplication.h>
#include <kpassdlg.h>
int main(int argc,char **argv)
{
   int code;
   QCString password;
   QString prompt = "Enter your password";
   KApplication app(argc,argv, "showpassworddialog");
   code = KPasswordDialog::getPassword(password,prompt);
   if(code == QDialog::Accepted) {
      // A password was entered
   } else if(code == QDialog::Rejected) {
      // A password was not entered
   }
   return(app.exec());
}
```
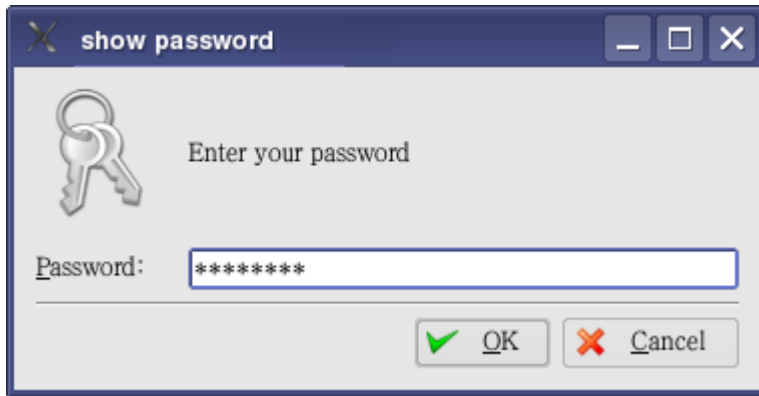
그림 19-39. 암호를 모르게 입력하는 KPasswordDialog창문부품

KPasswordEdit

KPasswordEdit창문부품은 암호로 사용할수 있도록 본문을 모르게 입력하는 단일행편집기이다.

머리부파일

#include <kpassdlg.h>

기초클라스

QLineEdit QObject QPaintDevice QWidget Qt

구성자

KPasswordEdit(QWidget *parent = 0, const char *name = 0);

메쏘드

void erase();

const char *password();

렬거형

enum EchoModes { OneStar, ThreeStars, NoEcho };

다음 실례는 그림 19-40에 보여주는 KPasswordEdit창문부품을 현시한다.

```
/* showpasswordedit.cpp */
#include <kapplication.h>
#include <kpassdlg.h>
int main(int argc,char **argv)
{
   KApplication app(argc,argv, "showpasswordedit");
   KPasswordEdit *passwordedit = new KPasswordEdit();
   passwordedit->show();
   app.setMainWidget(passwordedit);
```

```
    return(app.exec());
}
```

그림 19-40. 본문을 모르게 입력하는 KPasswordEdit창문부품

KPopupMenu

KPopupMenu창문부품은 KMenuBar 그리고 다른 KPopupMenu객체들과 결합하여 KDE표준차림표폼을 만드는데 사용된다. 이 창문부품은 QPopupMenu를 확장하여 마우스단추조종과 선택에 표준KDE환경을 사용하게 한다.

머리부파일

```
#include <kpopmenu.h>
```

기초클라스

QFrame QMenuData QObject QPaintDevice QPopupMenu QWidget Qt

구성자

```
KPopupMenu(QWidget *parent = 0, const char *name = 0);
KPopupMenu(const QString &title, QWidget *parent = 0, const char *name = 0);
```

메쏘드

```
void changeTitle(int id, const QString &text);
void changeTitle(int id, const QPixmap &icon, const QString &text);
int insertTitle(const QString &text, int id = - 1, int index = - 1);
int insertTitle(const QPixmap &icon, const QString &text, int id = - 1, int index = - 1);
void setTitle(const QString &title);
QString title(int id = - 1);
QPixmap titlePixmap(int id);
```

KMenuBar를 가진 KPopupMenu를 사용하는 실례는 이 장의 KMenuBar에 있다. 6장에 QPopupMenu를 사용하는 실례가 있다.

KProgress

KProgress창문부품은 수평 또는 수직으로 향하는 진척상황띠이다.

머리부파일

```
#include <kprogress.h>
```

기초클라스

QFrame QObject QPaintDevice QRangeControl QWidget Qt

구성자

```
KProgress(QWidget *parent = 0, const char *name = 0);
KProgress(Orientation, QWidget *parent = 0, const char *name = 0);
KProgress(int minValue, int maxValue, int value, Orientation, QWidget *parent = 0,
        const char *name = 0);
```

메쏘드

```
const QColor & barColor() const;
const QPixmap *barPixmap() const;
BarStyle barStyle() const;
QString format() const;
Orientation orientation() const;
void setBarColor(const QColor &);
void setBarPixmap(const QPixmap &);
void setBarStyle(BarStyle style);
void setFormat(const QString &format);
void setOrientation(Orientation);
void setTextEnabled(bool);
virtual QSize sizeHint() const;
virtual QSizePolicy sizePolicy() const;
bool textEnabled() const;
```

처리부

```
void advance(int prog);
void setValue(int);
```

신호

```
void percentageChanged(int);
```

렬거형

```
enum Orientation { Horizontal, Vertical };
enum BarStyle { Solid, Blocked };
```

다음 실례는 그림 19-41에 보여주는 수평KProgress창문부품을 현시한다. 진척상황띠의 모양과 형식을 바꾸는데 사용할수 있는 메쏘드는 많다.

```
/* showprogress.cpp */
#include <kapplication.h>
#include <kprogress.h>
int main(int argc,char **argv)
{
  KApplication app(argc,argv, "showprogress");
```

```
    KProgress *progress = new KProgress(0,500,350, KProgress::Horizontal);
    progress->show();
    app.setMainWidget(progress);
    return(app.exec());
}
```
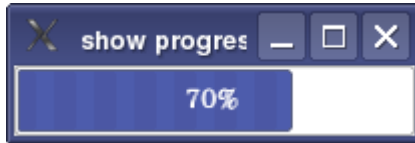
그림 19-41. 진행률을 보여주는 KProgress창문부품

KRestrictedLine

KRestrictedLine창문부품은 입력으로서 한행의 본문을 받아들이고 입력을 지정된 문자들의 모임으로 제한한다.

머리부파일

```
#include <krestrictedline.h>
```

기초클라스

KCompletionBase KLineEdit QLineEdit QObject QPaintDevice QWidget Qt

구성자

```
KRestrictedLine(QWidget *parent = 0, const char *name = 0, const QString &valid = QString::null);
```

메쏘드

```
void setValidChars(const QString &valid);
```

신호

```
void invalidChar(int);
```

다음 실례는 그림 19-42에 보여주는 KRestrictedLine창문부품을 현시한다. setValidChars() 호출은 대소문자모음과 대문자Y로 입력을 제한한다. 어떠한 다른 문자도 받아들이지 않고 invalidChar()신호를 발생시킨다.

```
/* showrestrictedline.cpp */
#include <kapplication.h>
#include <krestrictedline.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showrestrictedline");
    KRestrictedLine *restrictedline = new KRestrictedLine();
    restrictedline->setValidChars("aeiouAEIOUY");
    restrictedline->show();
```

```
    app.setMainWidget(restrictedline);
    return(app.exec());
}
```
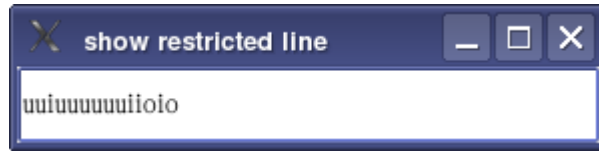
그림 19-42. 허용된 문자들만 보여주는 KRestricedLine창문부품

KRootPermsIcon

KRootPermissions창문부품은 현재사용자가 root허가를 가지는가를 가리키는 그림기호를 현시한다.

머리부파일

```
#include <kauthicon.h>
```

기초클라스

KAuthIcon QObject QPaintDevice QWidget Qt

구성자

```
KRootPermsIcon(QWidget *parent = 0, const char *name = 0);
```

메쏘드

```
bool status() const;
```

처리부

```
void updateStatus();
```

다음 실례는 제일 웃준위창문으로서 KRootPermsIcon창문부품을 사용하며 사용자가 root 허가를 가지는가 못가지는가에 따라 그림 19-43에서 보여주는 두 그림기호중의 하나를 현시한다.

```
/* showrootpermsicon.cpp */
#include <kapplication.h>
#include <kauthicon.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showrootpermsicon");
    KRootPermsIcon *rootpermsicon = new KRootPermsIcon();
    rootpermsicon->show();
    app.setMainWidget(rootpermsicon);
    return(app.exec());
```

}

그림 19-43. KRootPermsIcon창문부품현시의 2가지 형식

KRuler

KRuler창문부품은 눈금자처럼 표식이 붙은 수직 혹은 수평창문을 현시한다.

머리부파일

#include <kruler.h>

기초클라스

QFrame QObject QPaintDevice QWidget Qt

구성자

KRuler(KRuler::direction dir, QWidget *parent = 0, const char *name = 0, WFlags f = 0,
bool allowLines = TRUE);

KRuler(KRuler::direction dir, int widgetWidth, QWidget *parent = 0, const char *name = 0,
WFlags f = 0, bool allowLines = TRUE);

메쏘드

inline int getBigMarkDistance() const;

int getEndOffset() const;

int getLength() const;

bool getLengthFix() const;

inline int getLittleMarkDistance() const;

inline int getMaxValue() const;

inline int getMediumMarkDistance() const;

metric_style getMetricRulerStyle() const;

inline int getMinValue() const;

inline int getOffset() const;

paint_style getPaintRulerStyle() const;

inline double getPixelPerMark() const;

bool getShowBigMarks() const;

bool getShowEndMarks() const;

bool getShowLittleMarks() const;

bool getShowMediumMarks() const;

bool getShowPointer() const;

```
bool getShowTinyMarks() const;
paint_style getTickStyle() const;
inline int getTinyMarkDistance() const;
inline int getValue() const;
void setBigMarkDistance(int);
void setEndLabel(const QString &);
void setLength(int);
void setLengthFix(bool fix);
void setLittleMarkDistance(int);
void setMaxValue(int);
void setMediumMarkDistance(int);
void setMinValue(int);
void setOffset(int offset);
void setPixelPerMark(double);
void setRange(int min, int max);
void setRulerStyle(KRuler::metric_style);
void setRulerStyle(KRuler::paint_style);
void setTickStyle(KRuler::paint_style);
void setTinyMarkDistance(int);
void setValue(int);
void setValuePerBigMark(int);
void setValuePerLittleMark(int);
void setValuePerMediumMark(int);
void showBigMarkLabel(bool);
void showBigMarks(bool);
void showEndLabel(bool);
void showEndMarks(bool);
void showLittleMarkLabel(bool);
void showLittleMarks(bool);
void showMediumMarkLabel(bool);
void showMediumMarks(bool);
void showPointer(bool);
void showTinyMarks(bool);
void slidedown(int count = 1);
void slideup(int count = 1);
```

처리부

```
void slotEndOffset(int);
void slotNewOffset(int);
void slotNewValue(int);
```

렬거형

```
enum direction { horizontal, vertical };
enum metric_style { custom=0, pixel, inch, millimetres, centimetres, metres };
enum paint_style { flat, raised, sunken };
```

다음 실례프로그람은 그림 19-44에 보여주는 KRuler를 현시한다.

```
/* showruler.cpp */
#include <kapplication.h>
#include <kruler.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showruler");
    KRuler *ruler = new KRuler(KRuler::horizontal);
    ruler->setRulerStyle(KRuler::pixel);
    ruler->setLength(1000);
    ruler->setValue(750);
    ruler->showBigMarks(TRUE);
    ruler->showTinyMarks(FALSE);
    ruler->showMediumMarks(FALSE);
    ruler->showLittleMarks(FALSE);
    ruler->showEndMarks(FALSE);
    ruler->showEndLabel(FALSE);
    ruler->show();
    app.setMainWidget(ruler);
    return(app.exec());
}
```
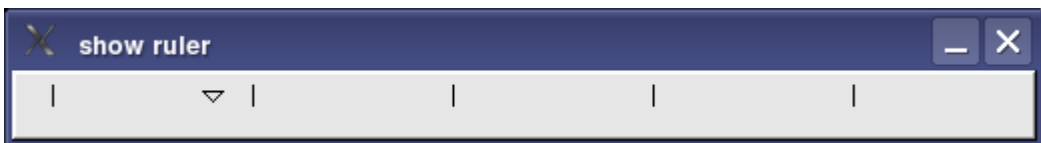
그림 19-44. 간단한 KRuler

KSelector

KSelector창문부품은 창문안의 수평 혹은 수직위치를 선택하는 창문부품을 창조하는데 쓰이는 기초클라스이다.

머리부파일

```
#include <kselect.h>
```

기초클라스

QObject QPaintDevice QRangeControl QWidget Qt

파생클라스

KGradientSelector KValueSelector

구성자

```
KSelector(Orientation o, QWidget *parent = 0L, const char *name = 0L);
```

메쏘드

```
QRect contentsRect();
bool indent() const;
Orientation orientation() const;
void setIndent(bool i);
```

신호

```
void valueChanged(int value);
```

렬거형

```
enum Orientation { Horizontal, Vertical };
```

다음 실례는 그림 19-45와 같이 수평KSelector창문부품을 현시한다. 창문바닥의 경계선에 있는 지시기는 마우스에 의해 이동할수 있으며 valueChanged()신호를 발생한다.

```
/* showselector.cpp */
#include <kapplication.h>
#include <kselect.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showselector");
    KSelector *selector = new KSelector(KSelector::Horizontal);
    selector->show();
    app.setMainWidget(selector);
    return(app.exec());
}
```
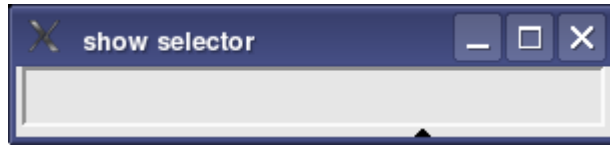
그림 19-45. 바닥에 선택지시기를 보여주는 수평KSelector창문부품

KSeparator

KSeparator창문부품은 표준KDE구분기창문부품이다. 론리그룹을 만들 때 차림표와 다른 창문에서 사용된다.

머리부파일

```
#include <kseparator.h>
```

기초클라스

QFrame QObject QPaintDevice QWidget Qt

구성자

```
KSeparator(QWidget *parent = 0, const char *name = 0, WFlags f = 0);
KSeparator(int orientation, QWidget *parent = 0, const char *name = 0, WFlags f = 0);
```

메쏘드

```
int orientation() const;
void setOrientation(int);
virtual QSize sizeHint() const;
```

다음 실례는 그림 19-46에 보여주는 수평KSeparator창문부품을 현시한다.

```
/* showseparator.cpp */
#include <kapplication.h>
#include <kseparator.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showseparator");
    KSeparator *separator = new KSeparator(Qt::Horizontal);
    separator->show();
    app.setMainWidget(separator);
    return(app.exec());
}
```
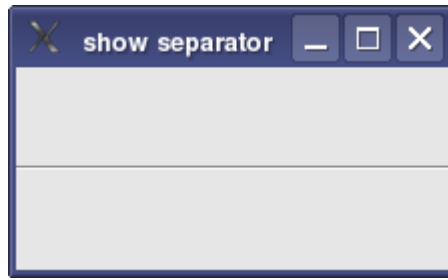
그림 19-46. 수평KSeparator창문부품

KSpellConfig

KSpellConfig창문부품은 철자검사대화칸을 현시한다.

머리부파일

```
#include <ksconfig.h>
```

기초클라스

QObject QPaintDevice QWidget Qt

구성자

```
KSpellConfig(QWidget *parent = 0, const char *name = 0, KSpellConfig *spellConfig = 0,
    bool addHelpButton = true);
KSpellConfig(const KSpellConfig &);
```

메쏘드

```
int client() const;
bool dictFromList() const;
const QString dictionary() const;
int encoding() const;
QStringList ignoreList() const;
bool noRootAffix() const;
bool runTogether() const;
void setClient(int client);
void setDictFromList(bool dfl);
void setDictionary(const QString qs);
void setEncoding(int enctype);
void setIgnoreList(QStringList _ignorelist);
void setNoRootAffix(bool);
void setRunTogether(bool);
bool writeGlobalSettings();
```

처리부

```
void activateHelp(void);
```

다음 실례는 제일 웃준위창문으로서 그림 19-47에 보여주는 KSpellConfig창문을 현시한다.

```
/* showspellconfig.cpp */
#include <kapplication.h>
#include <ksconfig.h>
int main(int argc,char **argv)
{
   KApplication app(argc,argv, "showspellconfig");
   KSpellConfig *spellconfig = new KSpellConfig();
   spellconfig->show();
   app.setMainWidget(spellconfig);
   return(app.exec());
}
```
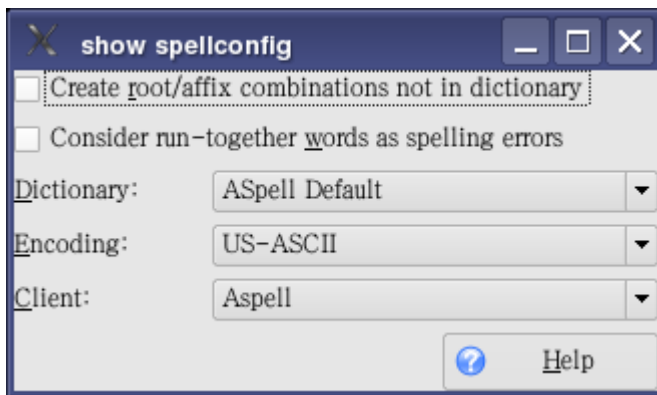
그림 19-47. KSpellConfig창문부품

KSpellDlg

KSpellDlg창문부품은 철자가 틀린 단어들과 수정안들의 목록을 현시하고 그 처리방법을 지정하는 단추들에 응답한다.

머리부파일

```
#include <kspelldlg.h>
```

기초클라스

QObject QPaintDevice QWidget Qt

구성자

```
KSpellDlg(QWidget *parent, const char *name, bool _progressbar = FALSE, bool _modal = FALSE);
```

메쏘드

```
void init(const QString &_word, QStringList *_sugg);
inline QString replacement();
void standby();
```

처리부

```
void slotProgress(unsigned int p);
```

신호

```
void command(int);
```

KSpellDlg의 다음 실례는 그림 19-48과 같이 철자가 틀린 단어들과 수정안들의 목록으로 현시를 초기화한다. init()메쏘드는 자료를 삽입하고 모든 단추들을 가능케 하며 standby()메쏘드는 단추들을 금지한다.

```
/* showspelldlg.cpp */
#include <kapplication.h>
#include <kspelldlg.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showspelldlg");
    QString word = "tehn";
    QStringList *suggestion = new QStringList();
    suggestion->append("the");
    suggestion->append("then");
    suggestion->append("ten");
    KSpellDlg *spelldlg = new KSpellDlg(0, "spelldlg");
    spelldlg->init(word,suggestion);
    spelldlg->show();
    app.setMainWidget(spelldlg);
    return(app.exec());
}
```

그림 19-48. KSpellDlg창문부품

KStatusBar

KStatusBar창문부품은 KMainWindow의 현시부분에 포함된다. 본문, 도형과 같은 상태를 현시하거나 사용자정의창문부품으로 사용된다.

머리부파일

```
#include <kstatusbar.h>
```

기초클라스

QObject QPaintDevice QStatusBar QWidget Qt

구성자

```
KStatusBar(QWidget *parent = 0L, const char *name = 0L);
```

메쏘드

```
void changeItem(const QString &text, int id);
inline void insertFixedItem(const QString &text, int ID, bool permanent = false);
void insertItem(const QString &text, int ID, int stretch = 0, bool permanent = false);
void removeItem(int id);
void setItemAlignment(int id, int align);
void setItemFixed(int id, int width = - 1);
```

신호

void pressed(int);

void released(int);

렬거형

enum BarStatus { Toggle, Show, Hide };

6장에 KStatusBar를 사용하는 실례가 있다.

KStatusBarLabel

KStatusBarLabel은 KStatusBar창문부품내에서 사용되며 마우스에 의해 조종되는 특별한 표식자이다.

머리부파일

#include <kstatusbar.h>

기초클라스

QFrame QLabel QObject QPaintDevice QWidget Qt

구성자

KStatusBarLabel(const QString &text, int _id, KStatusBar *parent = 0L, const char *name = 0L);

신호

void itemPressed(int id);

void itemReleased(int id);

다음 실례는 그림 19-49에서 보여주는 KStatusBarLabel창문부품을 창조하고 현시한다. 구성자는 표식자본문뿐아니라 신호와 함께 제공되는 식별번호도 가진다.

```
/* showstatusbarlabel.cpp */
#include <kapplication.h>
#include <kstatusbar.h>
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showstatusbarlabel");
    KStatusBarLabel *statusbarlabel = new KStatusBarLabel("Label text",4);
    statusbarlabel->show();
    app.setMainWidget(statusbarlabel);
    return(app.exec());
}
```
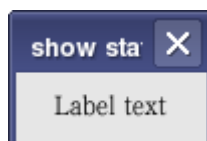
그림 19-49. KStatusBarLabel창문부품

KTextBrowser

KTextBrowser창문부품은 URL과 전자우편련결을 인식하고 응답하도록 추가기능을 가지는 본문열람기이다.

머리부파일

```
#include <ktextbrowser.h>
```

기초클라스

QFrame QObject QPaintDevice QScrollView QTextBrowser QTextView QWidget Qt

구성자

```
KTextBrowser(QWidget *parent = 0, const char *name = 0, bool notifyClick = false);
```

메쏘드

```
void setNotifyClick(bool notifyClick);
```

신호

```
void mailClick(const QString &name, const QString &address);
void urlClick(const QString &url);
```

다음 실례는 그림 19-5 0 에 보여주는 본문을 현시한다. URL과 전자우편응답은 setNotifyClick()를 호출하여 능동으로 혹은 해제될수 있다.

```
/* showtextbrowser.cpp */
#include <kapplication.h>
#include <ktextbrowser.h>
char text[] =
    "This is the text being displayed\n"
    "by the text browser. Both vertical\n"
    "and horizontal scroll bars will\n"
    "appear as necessary. ";
int main(int argc,char **argv)
{
    KApplication app(argc,argv, "showtextbrowser");
    KTextBrowser *textbrowser = new KTextBrowser();
    textbrowser->show();
    textbrowser->setText(QString(text));
    app.setMainWidget(textbrowser);
    return(app.exec());
}
```
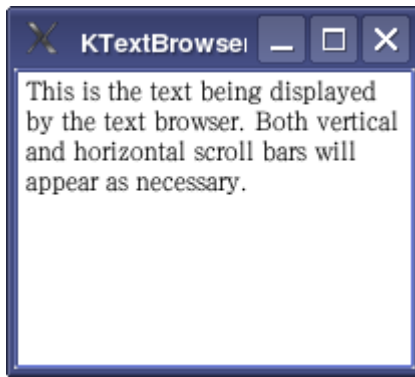
그림 19-50. 본문을 현시하는 KTextBrowswer창문부품

KToolBar

KToolBar창문부품은 몇가지 종류의 도구창문부품들을 포함할수 있고 여러 위치에서 끌고다닐수 있는 도구띠이다.

머리부파일

```
#include <ktoolbar.h>
```

기초클라스

QFrame QObject QPaintDevice QWidget Qt

파생클라스

KFormulaToolBar

구성자

```
KToolBar(QWidget *parent = 0L, const char *name = 0L, bool _honor_mode = false);
```

메쏘드

```
void addConnection(int id, const char *signal, const QObject *receiver, const char *slot);
void alignItemRight(int id, bool right = true);
KAnimWidget *animatedWidget(int id);
BarPosition barPos() const;
void changeComboItem(int id, const QString &text, int index = - 1);
void clear();
void clearCombo(int id);
bool contextMenuEnabled() const;
int count();
bool enable(BarStatus stat);
void enableFloating(bool arrrrrrgh);
void enableMoving(bool flag = true);
bool fullSize() const;
```

```
KToolBarButton *getButton(int id);
QComboBox *getCombo(int id);
QString getComboItem(int id, int index = - 1);
KLineEdit *getLined(int id);
QString getLinedText(int id);
QWidget *getWidget(int id);
virtual int heightForWidth(int width) const;
void hideItem(int id);
int iconSize() const;
IconText iconText() const;
int insertAnimatedWidget(int id, QObject *receiver, const char *slot, const QStringList &icons,
        int index = - 1);
int insertButton(const QString &icon, int id, bool enabled = true, const QString &text = QString::null,
        int index = - 1);
int insertButton(const QString &icon, int id, const char *signal, const QObject *receiver,
        const char *slot, bool enabled = true, const QString &text = QString::null, int index = - 1);
int insertButton(const QPixmap &pixmap, int id, bool enabled = true,
        const QString &text = QString::null, int index = - 1);
int insertButton(const QPixmap &pixmap, int id, const char *signal, const QObject *receiver,
        const char *slot, bool enabled = true, const QString &text = QString::null, int index = - 1);
int insertButton(const QPixmap &pixmap, int id, QPopupMenu *popup, bool enabled,
        const QString &_text, int index = - 1);
int insertCombo(QStrList *list, int id, bool writable, const char *signal, const QObject *receiver,
        const char *slot, bool enabled = true, const QString &tooltiptext = QString::null, int size = 70,
        int index = - 1, QComboBox::Policy policy = QComboBox::AtBottom);
int insertCombo(const QStringList &list, int id, bool writable, const char *signal,
        const QObject *receiver, const char *slot, bool enabled = true,
        const QString &tooltiptext = QString::null, int size = 70, int index = - 1,
        QComboBox::Policy policy = QComboBox::AtBottom);
int insertCombo(const QString &text, int id, bool writable,
        const char *signal, QObject *recevier, const char *slot, bool enabled = true,
        const QString &tooltiptext = QString::null, int size = 70, int index = - 1,
        QComboBox::Policy policy = QComboBox::AtBottom);
void insertComboItem(int id, const QString &text, int index);
void insertComboList(int id, QStrList *list, int index);
void insertComboList(int id, const QStringList &list, int index);
```

```
int insertLineSeparator(int index = - 1);
int insertLined(const QString &text, int ID, const char *signal, const QObject *receiver,
        const char *slot, bool enabled = true, const QString &toolTipText = QString::null, int size = 70,
        int index = - 1);
int insertSeparator(int index = - 1);
int insertWidget(int id, int width, QWidget *_widget, int index = - 1);
bool isButtonOn(int id);
int maxHeight();
int maxWidth();
virtual QSize maximumSizeHint() const;
virtual QSize minimumSizeHint() const;
void removeComboItem(int id, int index);
void removeItem(int id);
void saveState();
void setAutoRepeat(int id, bool flag = true);
void setBarPos(BarPosition bpos);
void setButton(int id, bool flag);
void setButtonIcon(int id, const QString &_icon);
void setButtonPixmap(int id, const QPixmap &_pixmap);
void setCurrentComboItem(int id, int index);
void setDelayedPopup(int id, QPopupMenu *_popup, bool toggle = false);
void setEnableContextMenu(bool enable = true);
void setFlat(bool flag);
void setFullSize(bool flag = true);
void setIconSize(int size);
void setIconSize(int size, bool update);
void setIconText(IconText it);
void setIconText(IconText it, bool update);
void setItemAutoSized(int id, bool yes = true);
void setItemEnabled(int id, bool enabled);
void setItemNoStyle(int id, bool no_style = true);
void setLinedText(int id, const QString &text);
void setMaxHeight(int h);
void setMaxWidth(int dw);
void setTitle(const QString &_title);
void setToggle(int id, bool flag = true);
```

```
void setXML(const QString &xmlfile, const QDomDocument &xml);
void showItem(int id);
virtual QSize sizeHint() const;
virtual QSizePolicy sizePolicy() const;
void toggleButton(int id);
void updateRects(bool resize = false);
virtual int widthForHeight(int height) const;
```

신호

```
void clicked(int id);
void doubleClicked(int id);
void highlighted(int id, bool isHighlighted);
void highlighted(int id);
void modechange();
void moved(BarPosition);
void pressed(int);
void released(int);
void toggled(int);
```

렬거형

```
enum IconText { IconOnly=0, IconTextRight, TextOnly, IconTextBottom };
enum BarStatus { Toggle, Show, Hide };
enum BarPosition { Top=0, Left, Right, Bottom, Floating, Flat };
```

6장에 KToolBar창문부품의 실례가 있다.

KToolBarButton

KToolBarButton창문부품은 마우스사건에 응답하는 단추들을 현시하기 위하여 KToolBar창문부품에 의해 내부적으로 사용된다. KToolBarButton은 KToolBar의 insertButton()메쏘드들중 하나를 호출하여 만든다. 식별번호를 사용한 KToolBar메쏘드 getButton()호출은 단추의 지적자를 얻어서 목록에 있는 메쏘드들을 사용가능하게 한다.

머리부파일

```
#include <ktoolbarbutton.h>
```

기초클라스

QButton QObject QPaintDevice QWidget Qt

구성자

```
KToolBarButton(const QString &icon, int id, QWidget *parent, const char *name = 0L,
        const QString &txt = QString::null);
```

```
KToolBarButton(const QPixmap &pixmap, int id, QWidget *parent, const char *name = 0L,
        const QString &txt = QString::null);
KToolBarButton(QWidget *parent = 0L, const char *name = 0L);
```

메쏘드

```
void on(bool flag = true);
QPopupMenu *popup();
virtual void setDefaultIcon(const QString &icon);
virtual void setDefaultPixmap(const QPixmap &pixmap);
void setDelayedPopup(QPopupMenu *p, bool toggle = false);
virtual void setDisabledIcon(const QString &icon);
virtual void setDisabledPixmap(const QPixmap &pixmap);
void setEnabled(bool enable = true);
virtual void setIcon(const QString &icon);
virtual void setIcon(const QString &icon, bool generate);
void setNoStyle(bool no_style = true);
virtual void setPixmap(const QPixmap &pixmap);
virtual void setPixmap(const QPixmap &pixmap, bool generate);
void setPopup(QPopupMenu *p);
void setRadio(bool f = true);
virtual void setText(const QString &text);
void setToggle(bool toggle = true);
void toggle();
```

처리부

```
void modeChange();
```

신호

```
void clicked(int);
void doubleClicked(int);
void highlighted(int, bool);
void pressed(int);
void released(int);
void toggled(int);
```

6장의 KToolBar에 KToolBarButton창문부품들을 만드는 실례들이 있다.

KWizard

KWizard창문부품은 단계에 따라서 사용자를 차림표하는 대화칸을 만드는데 쓰인다. 매

개 단계는 한개 창문으로 이루어진다. KWizard창문부품은 페지작성도구와 조종단추들을 제공한다. 이 창문부품은 QWizard를 확장하여 대화칸에 표준KDE메쏘드모임을 포함하게 한다. 그것은 KDialogBase를 구축하는 기초클라스이다.

머리부파일

#include <kwizard.h>

기초클라스

QDialog QObject QPaintDevice QWidget QWizard Qt

구성자

KWizard(QWidget *parent = 0, const char *name = 0, bool modal = false, WFlags f = 0);

다음 실례는 그림 19-51에 보여주는 빈 KWizard창문부품을 현시한다.

```
/* showwizard.cpp */
#include <kapplication.h>
#include <kwizard.h>
int main(int argc,char **argv)
{
    QApplication app(argc,argv);
    KWizard *wizard = new KWizard();
    wizard->show();
    app.setMainWidget(wizard);
    return(app.exec());
}
```
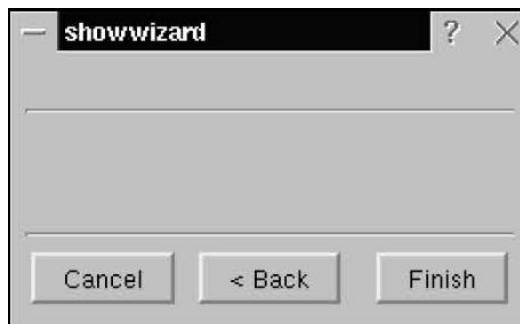
그림 19-51. KWizard창문부품

KXYSelector

KXYSelector창문부품은 QWidget대신에 쓰이는 기초클라스이고 마우스로 표면우의 점을 선택하는 기능을 추가한다.

머리부파일

#include <kselect.h>

기초클라스

QObject QPaintDevice QWidget Qt

파생클라스

KHSSelector

구성자

```
KXYSelector(QWidget *parent = 0L, const char *name = 0L);
```

메쏘드

```
QRect contentsRect();
void setRange(int _minX, int _minY, int _maxX, int _maxY);
void setValues(int _xPos, int _yPos);
int xValue();
int yValue();
```

신호

```
void valueChanged(int _x, int _y);
```

KXZWidget의 다음 실례는 그림 19-52에 보여주는 +위치지시기를 현시한다.

```
/* showxyselector.cpp */
#include <kapplication.h>
#include <kselect.h>
int main(int argc,char **argv)
{
    QApplication app(argc,argv);
    KXYSelector *xyselector = new KXYSelector();
    xyselector->show();
    app.setMainWidget(xyselector);
    return(app.exec());
}
```

그림 19-52. +위치지시기를 가지는 KXYSelector창문부품

요 약

이 장은 매개 KDE창문부품들의 자모순목록을 제공하였다. 매개 창문부품은 다음과 같은 형식으로 구성되였다.

· 창문부품의 실례들을 만드는데 쓰이는 구성자들

· 창문부품이 정의되는.머리부파일의 이름

· 창문부품이 기능을 계승하는 모든 기초클라스들

· 창문부품이 기능을 넘겨주는 모든 파생클라스들

· 1개 창문부품의 사건을 다른 창문부품의 메쏘드호출에 련결하는데 사용되는 처리부와 신호들

· 응용프로그람에 사용할수 있는 공개메쏘드

이 장과 앞장은 KDE환경에서 실행하는 프로그람의 실례를 많이 포함하였다. 이 프로그람과 다른 체계용으로 쓴 프로그람들사이에 차이가 많고 류사점도 많다. 다음 장은 간단한 KDE프로그람과 Windows용으로 작성한 같은 프로그람의 기본구조를 하나씩 비교한다.

제20장. 창문프로그람의 비교분석

학습내용

창문에 바른4각형들을 그리는 간단한 Win32프로그람의 작성

창문에 바른4각형들을 그리는 간단한 KDE프로그람의 작성

창문에 바른4각형들을 그리는 간단한 GNOME프로그람의 작성

KDE와 Win32프로그람의 비교

이미 Win32 API를 사용한 프로그람작성에 익숙되여있으면 이 장은 KDE/Qt응용프로그람의 구조를 리해하는데 도움이 될것이다. 가장 낮은 준위에서 두 프로그람작성모형은 매우 류사하다. 그것들은 모두 사건들이 들어오기를 기다리는 기본순환고리를 리용하여 조작하며 사건이 도달할 때 함수는 응용프로그람에 통지하기 위하여 호출된다.
될수록 간단히 비교하기 위하여 이 장은 Win32와 KDE모두를 위한 간단한 프로그람을 실현한다.

제1절. Win32프로그람

다음의 실례는 중심이 같은 직4각형들로 창문을 가득 채우는 Windows프로그람이다. 창문크기가 변할 때마다 직4각형들도 그것에 맞게 크기가 달라진다. 결과창문은 그림 20-1과 같다.

BoxBox

```
1 /* boxbox.c (win32) */
2 #include <windows.h>
3
4 #define STEP 3
5
6 static char name[] = “BoxBox”;
7 static int xBox1;
8 static int yBox1;
9 static int xBox2;
10 static int yBox2;
11
12 LRESULT CALLBACK callback(HWND,UINT,WPARAM,LPARAM);
13
14 int WINAPI WinMain(HINSTANCE instance,
15     HINSTANCE prev,PSTR commandLine,int showCommand)
16 {
```

```
17    HWND window;
18    MSG message;
19    WNDCLASSEX winclass;
20
21    winclass.cbSize = sizeof (winclass);
22    winclass.style = CS_HREDRAW | CS_VREDRAW;
23    winclass.lpfnWndProc = callback;
24    winclass.cbClsExtra = 0;
25    winclass.cbWndExtra = 0;
26    winclass.hInstance = instance;
27    winclass.hIcon = LoadIcon(NULL,IDI_APPLICATION);
28    winclass.hCursor = LoadCursor(NULL,IDC_ARROW);
29    winclass.lpszMenuName = NULL;
30    winclass.lpszClassName = name;
31    winclass.hIconSm = LoadIcon(NULL,IDI_APPLICATION);
32    winclass.hbrBackground =
33          (HBRUSH)GetStockObject(WHITE_BRUSH);
34
35    RegisterClassEx(&winclass);
36    window = CreateWindow (name,”Boxes in Boxes”,
37          WS_OVERLAPPEDWINDOW,
38          CW_USEDEFAULT,CW_USEDEFAULT,
39          CW_USEDEFAULT,CW_USEDEFAULT,
40          NULL,NULL,instance,NULL);
41    ShowWindow(window,showCommand);
42    UpdateWindow (window);
43
44    while(GetMessage(&message,NULL,0,0)) {
45          TranslateMessage(&message);
46          DispatchMessage(&message);
47    }
48    return(message.wParam);
49 }
50
51 LRESULT CALLBACK callback(HWND window,UINT messageType,
```

```
52    WPARAM wParam,LPARAM lParam)
53 {
54    int x1;
55    int y1;
56    int x2;
57    int y2;
58    HDC hdc;
59    PAINTSTRUCT ps;
60
61    switch (messageType) {
62            case WM_SIZE:
63                    xBox1 = 10;
64                    yBox1 = 10;
65                    xBox2 = LOWORD(lParam) - 10;
66                    yBox2 = HIWORD(lParam) - 10;
67                    return(0);
68            case WM_PAINT:
69                    hdc = BeginPaint(window,&ps);
70                    SetViewportOrgEx(hdc,0,0,NULL);
71                    x1 = xBox1;
72                    x2 = xBox2;
73                    y1 = yBox1;
74                    y2 = yBox2;
75                    while((x1 < x2) && (y1 < y2)) {
76                            MoveToEx(hdc,x1,y1,NULL);
77                            LineTo(hdc,x2,y1);
78                            LineTo(hdc,x2,y2);
79                            LineTo(hdc,x1,y2);
80                            LineTo(hdc,x1,y1);
81                            x1 += STEP;
82                            y1 += STEP;
83                            x2 -= STEP;
84                            y2 -= STEP;
85                    }
86                    EndPaint(window,&ps);
```

```
87              return(0);
88          case WM_DESTROY:
89              PostQuitMessage(0);
90              return  (0);
91      }
92      return(DefWindowProc(window,messageType,
93          wParam,lParam));
94 }
```
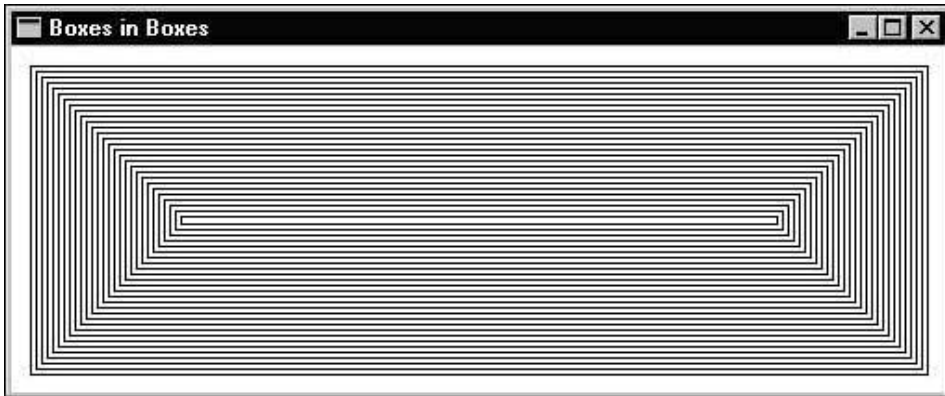
그림 20-1. Win32기본창문

프로그람은 2개 함수만 가지고있다. 14행에서 시작하는 함수 WinMain()은 프로그람을 시작하기 위하여 조작체계에 의해 호출되는 함수이다. 51행의 함수 callback()는 사건이 일어날 때마다 조작체계에 의해 호출된다.

제2절. KDE프로그람

다음 실례는 중심이 같은 직4각형들로 창문을 가득 채우는 KDE프로그람이다. 창문크기가 변할 때마다 직4각형들도 그것에 맞게 크기가 달라진다. 창문은 그림 20-2와 같다. 실례는 2개의 파일 즉 BoxBox창문부품을 선언하는 머리부파일과 프로그람의 기본함수와 BoxBox클라스의 메쏘드들의 본체들을 포함하는 C++원천파일로 되여있다.

BoxBox머리부파일

```
1 /* boxbox.h (KDE) */
2 #ifndef BOXBOX_H
3 #define BOXBOX_H
4
5 #include <qwidget.h>
6
7 class BoxBox: public QWidget
```

```
8 {
9 public:
10     BoxBox(QWidget *parent=0,const char *name=0);
11 private:
12     int xBox1;
13     int yBox1;
14     int xBox2;
15     int yBox2;
16 protected:
17     virtual void paintEvent(QPaintEvent *);
18     virtual void resizeEvent(QResizeEvent *);
19 };
20
21 #endif
```

BoxBox

```
1 /* boxbox.cpp (KDE) */
2 #include <kapplication.h>
3 #include <qpainter.h>
4 #include “boxbox.h”
5
6 #define STEP 3
7
8 int main(int argc,char **argv)
9 {
10     KApplication app(argc,argv,”boxbox”);
11     BoxBox boxbox;
12     boxbox.show();
13     app.setMainWidget(&boxbox);
14     return(app.exec());
15 }
16 BoxBox::BoxBox(QWidget *parent,const
17     char *name) : QWidget(parent,name)
18 {
19     resize(400,200);
20 }
```

```
21 void BoxBox::paintEvent(QPaintEvent *)
22 {
23     QPainter p;
24
25     int x1 = xBox1;
26     int y1 = yBox1;
27     int x2 = xBox2;
28     int y2 = yBox2;
29
30     p.begin(this);
31     while((x1 < x2) && (y1 < y2)) {
32             p.moveTo(x1,y1);
33             p.lineTo(x2,y1);
34             p.lineTo(x2,y2);
35             p.lineTo(x1,y2);
36             p.lineTo(x1,y1);
37             x1 += STEP;
38             y1 += STEP;
39             x2 -= STEP;
40             y2 -= STEP;
41     }
42     p.end();
43 }
44 void BoxBox::resizeEvent(QResizeEvent *e)
45 {
46     QSize size = e->size();
47     xBox1 = 10;
48     yBox1 = 10;
49     xBox2 = size.width() - 10;
50     yBox2 = size.height() - 10;
51 }
```
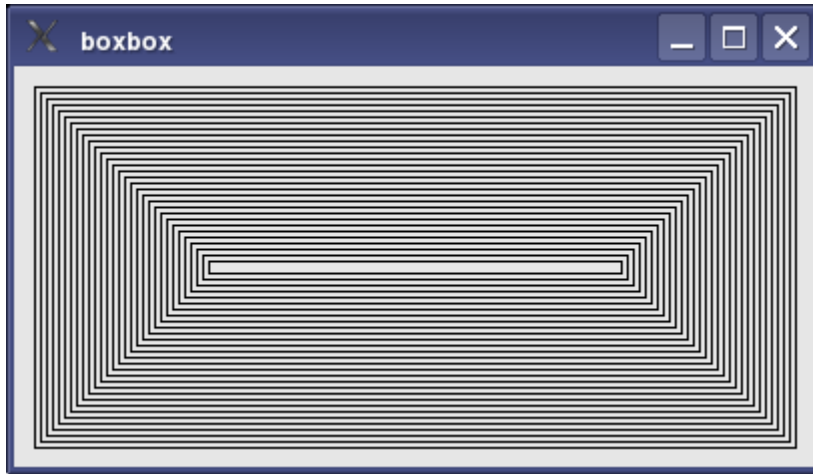
그림 20-2. KDE최상위창문

이 프로그람은 KDE환경을 초기화하고 창문부품을 만들어 현시하며 최상위창문으로 창문부품을 설정한다. 그리고 실행순환고리에 들어가는데 쓰이는 main()함수를 가지고있다. 구성자외에 특정한 사건이 발생할 때 실행되는 2개의 메쏘드가 있다.

제3절. Win32와 KDE의 대비

이 절은 KDE와 Win32프로그람들사이의 류사성과 차이점을 자세히 설명한다.

1. 초기화

KDE프로그람의 10행은 KApplication객체를 구축한다. 또한 기초하는 도형방식쏘프트웨어의 GUI대면부와 그밖의 부분들을 초기화한다. Win32 API가 조작체계의 부분이고 이미 초기화되기때문에 Win32에는 이 함수에 대응하는 부분이 없다. 그러나 KDE는 조작체계의 부분이 아니므로 창문쏘프트웨어를 초기화하여야 한다.

KApplication객체의 구성은 그 이름을 프로그람에 할당하며 이것은 기정적으로 창문의 제일 우에 나타나는 제목으로 사용된다. Win32판에서 같은 동작은 36행의 CreateWindow()호출이다.

2. 기본창문

KDE프로그람의 11행은 BoxBox객체를 만드는데 그것은 응용프로그람의 최상위창문으로 된다. Win32판의 21～33행은 기본창문을 정의하며 35행의 RegisterClassEx()호출은 그것을 조작체계에 등록한다. Win32최상위창문을 창조할 때 KDE보다 더 많은 환경설정이 요구된다. 이것은 Win32에서는 미리 모든 환경설정이 구체적으로 지정되여야 하고 KDE에서는 그 모두에 기정값들의 표준모임을 리용하고 창문이 구축된 후에 프로그람이 변경하는데 사용할 수 있는 함수들을 가지고있기때문이다.

3. 사건에 응답하기

KDE와 Win32프로그람은 모두 사건구동형이다. 다시 말하면 두 프로그람이 초기화되고 창문이 현시되면 KDE와 Win32는 사건(마우스, 건반 또는 기타)이 일어나기를 기다린다. 사

건이 발생할 때 체계에서 응용프로그람에로 정보를 넘기기 위한 함수가 호출된다. 두 프로그람은 2개의 특정한 사건 즉 창문크기가 변화할 때 그리고 창문의 전부 또는 일부분이 로출되여 다시 그려야 할 때 응답할 필요가 있다.

21행에서 KDE프로그람은 창문을 그려야 할 때마다 호출하는 QWidget의 순수가상메쏘드 paintEvent()를 재정의한다. 이 함수는 resizeEvent()의 최근호출로부터 얻어진 크기와 위치값들을 리용하여 겹쌓인 4각형들을 그린다. Win32프로그람은 68행에서 case문에 의해 이것을 달성한다.

KDE프로그람은 44행에서 창문이 처음으로 표시되고 그 크기가 바뀔 때마다 호출되는 QWidget의 가상메쏘드 resizeEvent()를 재정의한다. 같은 기능은 Win32프로그람의 callback()함수의 62행의 case문에서 달성된다. 두 경우에 새로운 크기정보는 xBox1, yBox1, xBox2, yBox2변수들에 기억된다.

알아두기: 사건을 처리하는 이 두 메쏘드는 류사하다. KDE응용프로그람은 매개 사건에 대하여 각이한 역호출메쏘드를 지정한다. Win32프로그람에서 1개 역호출함수는 매개 사건에 대하여 개별적인 case문을 포함하는 발송자로서 리용될수 있으므로 매개 사건에 대하여 개별적인 함수를 호출한다.

4. 기본순환고리

KDE프로그람은 14행의 exec()를 호출한다. 이 함수는 프로그람이 완료할 시간이 될 때까지 되돌아오지 않는다. 그것은 사건을 기다리다가 적당한 메쏘드를 호출하게 하는 일감을 수행한다. Win32프로그람의 기본순환고리는 44～47행에 있다. 함수 GetMessage()는 사건을 수신하기를 기다리며 그다음 되돌아간다. TranslateMessage()호출은 건반코드를 문자로 변환하고 DispatchMessage()는 정확한 창문에 사건을 보낸다.

5. 프로그람완료

Win32프로그람을 완료할 때 역호출함수는 WM_DESTROY통보문과 함께 호출된다. PostQuitMessage()호출은 89행에서 이루어지고 입력대기렬에 완료통보문을 배치하고 그것을 읽을 때 프로그람은 끝난다. 이것은 작성자에게 완료하기전에 모든 객체들을 삭제하거나 또는 완료를 거절할 기회를 준다.

KDE응용프로그람완료는 KApplication객체의 초기화과정에 설정되는 환경으로 구축된다. 최상위창문을 닫으면 응용프로그람도 완료한다. 또한 응용프로그람완료는 최상위창문을 닫는다. 창문꼭대기의 제목띠는 마우스가 창문을 닫는데 쓰일수 있도록 완료과정과 련결된다.

6. 대역자료

Win32프로그람은 C로 작성되였기때문에 그것은 역호출함수호출들사이에 남아있는 값들을 보관하는데 대역기억기를 사용할것을 요구한다. 물론 Win32판은 C++로 작성될수 있고 대역자료를 밀봉할수 있는 객체가 제안되였으나 그 수법은 API 자체의 부분이 아니다. 이 실례에서 대역자료는 7행에서 변수 xBox1, yBox1, xBox2, yBox2에 기억된다.

C++로 작성된 KDE는 매개의 개별적인 클라스안에 자료를 보관한다. 이 실례에서 제일 바깥쪽 직4각형의 모서리들은 xBox1, yBox1, xBox2, yBox2에 보관되며 머리부파일의 12～15행에서 정의되였다.

제4절. GNOME프로그람

다음 실례는 중심이 같은 직4각형들로 창문을 가득 채운다. 이전 실례처럼 창문크기가 변화하면 직4각형들의 크기도 변한다. 창문은 그림 20-3과 같다.

```
1 /** boxbox.c (Gnome) **/
2 #include <gnome.h>
3
4 gint eventDelete(GtkWidget *widget,
5       GdkEvent *event,gpointer data);
6 gint eventDestroy(GtkWidget *widget,
7       GdkEvent *event,gpointer data);
8 gboolean eventExpose(GtkWidget *widget,
9       GdkEvent *event,gpointer data);
10 gint eventConfigure(GtkWidget *widget,
11      GdkEventConfigure *event,gpointer data);
12
13 #define STEP 3
14
15 static char name[] = “BoxBox”;
16 static int xBox1;
17 static int yBox1;
18 static int xBox2;
19 static int yBox2;
20
21 int main(int argc,char *argv[])
22 {
23      GtkWidget *app;
24      GtkWidget *area;
25
26      gnome_init(name,”1.0”,argc,argv);
27      app = gnome_app_new(name,”Boxes in Boxes”);
28      gtk_signal_connect(GTK_OBJECT(app),”delete_event”,
```

```
29          GTK_SIGNAL_FUNC(eventDelete),NULL);
30     gtk_signal_connect(GTK_OBJECT(app),” destroy” ,
31          GTK_SIGNAL_FUNC(eventDestroy),NULL);
32
33     area = gtk_drawing_area_new();
34     gnome_app_set_contents(GNOME_APP(app),area);
35
36     gtk_signal_connect(GTK_OBJECT(area),” expose_event” ,
37          GTK_SIGNAL_FUNC(eventExpose),NULL);
38     gtk_signal_connect(GTK_OBJECT(area),” configure_event” ,
39          GTK_SIGNAL_FUNC(eventConfigure),NULL);
40
41     gtk_widget_show_all(app);
42     gtk_main();
43     exit(0);
44 }
45 gboolean eventExpose(GtkWidget *widget,
46          GdkEvent *event,gpointer data) {
47     int x1;
48     int y1;
49     int x2;
50     int y2;
51
52     x1 = xBox1;
53     y1 = yBox1;
54     x2 = xBox2;
55     y2 = yBox2;
56     while((x1 < x2) && (y1 < y2)) {
57          gdk_draw_line(widget->window,
58               widget->style->black_gc,
59               x1,y1,x2,y1);
60          gdk_draw_line(widget->window,
61               widget->style->black_gc,
62               x2,y1,x2,y2);
63          gdk_draw_line(widget->window,
```

```
64                  widget->style->black_gc,
65                  x2,y2,x1,y2);
66            gdk_draw_line(widget->window,
67                  widget->style->black_gc,
68                  x1,y2,x1,y1);
69            x1 += STEP;
70            y1 += STEP;
71            x2 -= STEP;
72            y2 -= STEP;
73      }
74      return (TRUE);
75 }
76 gint eventConfigure(GtkWidget *widget,
77      GdkEventConfigure *event,gpointer data)
78 {
79      xBox1 = 10;
80      yBox1 = 10;
81      xBox2 = event->width - 10;
82      yBox2 = event->height - 10;
83      return (TRUE);
84 }
85 gint eventDelete(GtkWidget *widget,
86            GdkEvent *event,gpointer data) {
87      return(FALSE);
88 }
89 gint eventDestroy(GtkWidget *widget,
90            GdkEvent *event,gpointer data) {
91      gtk_main_quit();
92      return (0);
93 }
```
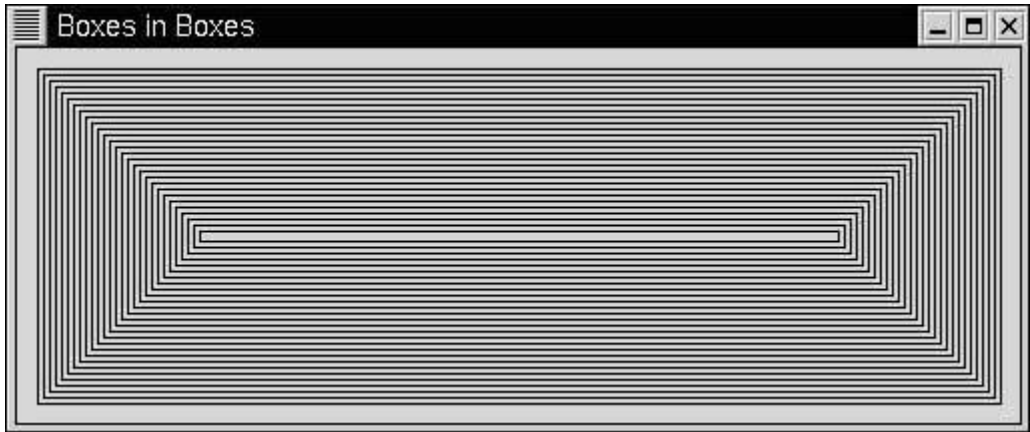
그림 20-3. GNOME기본창문

이 프로그람은 GNOME를 초기화하고 현시할 창문을 만들며 역호출들을 지정한다. 그리고 실행순환고리에 들어가는 main()을 가지고있다. 4가지의 다른 함수가 있는데 그 매개는 특정한 사건이 일어날 때 실행한다.

사건들에 응답하기 위하여 설정된것들을 얻으려면 좀 더 노력하여야 한다. 36행과 38행의 gtk_signal_connect()호출은 프로그람이 창문크기를 설정하고 그리는 사건들을 받는데 필요하다. 또한 응용프로그람이 사용자에 의해 정지될수 있도록 마우스에 응답하기 위한 역호출을 설정하는데도 필요하다. 이것은 28행과 30행에서 gtk_signal_connect()호출에 의해 수행된다.

요 약

Win32응용프로그람과 KDE응용프로그람의 기본구성방식은 같다. 모두 사건에 기초한다. GNOME응용프로그람도 류사한 특성을 가지고있다.

이 장에서는 다음에 같은것을 고찰하였다.

· Win32, KDE 그리고 GNOME은 모두 사건들의 도착을 기다리는 무한순환고리라는 개념을 사용한다.

· 그것들은 모두 같은 량의 코드를 요구하지만 어떤것은 다른것보다 특별한 위치에서 더 자세한것을 요구할수 있다.

· 3개의 응용프로그람은 모두 최상위창문이라는 개념을 공유하며 모든 최상위창문은 사건들을 받아들일수 있다.

· 그것들은 모두 응용프로그람이 정확히 닫기도록 하는 메쏘드를 제공한다.

부록. 쏘프트웨어개발을 위한 설정

KDE/Qt응용프로그람을 작성하기 전에 많은 쏘프트웨어 구성요소들을 설치하여야 한다.

1. Linux

우선 Linux CD를 구입해야 한다. 상업용CD는 필요한 대부분의 항목을 포함하고있으므로 내리적재판보다 설치하기 쉽다.

인터네트로 Linux문서를 얻으려고 한다면 판선택에 주의해야 한다. 주로 사용자들은 시험판을 얻으려고 하지 않는다. Linux판번호는 안정판과 개발판을 구분한다. Linux핵심자체에서 일하고있지 않는 한 안정판을 사용하여야 한다. 판번호뒤의 둘째 수자가 짝수(2.0, 2.2, …)이면 안정판이다. 판번호에서 3번째 수(2.2.8)는 오유고착상태를 가리킨다. 최근판의 핵심을 얻으려면 가장 큰 짝수번호판을 구입하여야 한다.

2. gcc(egcs)

이것은 프로그람을 콤파일하는데 필요한 C콤파일러이다. 1999년 4월에 앞으로의 계획때문에 Free Software Foundation는 egcs Steering Committeeto에 gcc을 유지할것을 확정하였다. 그당시 이름은 gcc에서 egcs로 바뀌였다. 그러나 모든 현존제작파일들과 작업해야 하므로 여전히 실행프로그람은 gcc라고 부른다.

Linux배포물에는 gcc콤파일러가 반드시 있다. 그러나 새로운 특성들이 때때로 추가되므로 현재 리용되고있는 KDE쏘프트웨어의 판을 가지고있는가 확인하는것이 필요하다.

3. XFree86

이것은 Qt에 의해 사용되는 저수준창문작성쏘프트웨어이다. 모든 Linux배포물에는 이 쏘프트웨어가 있으며 표준설치로 설치할수 있다. 이것은 매우 안정된 쏘프트웨어이지만 특수한 경우에만 사용하는 배포물이다.

4. autoconf와 automake

GNOME배포물 원천코드를 콤파일한다면 이 편의프로그람들이 필요된다. 이 편의프로그람들은 제작파일들을 자동적으로 만들고 프로그람을 콤파일하는데 사용된다. 콤파일과정에 문제가 제기되면 최종판을 가지고 확인해야 한다. 프로그람을 콤파일할 때 이 편의프로그람들을 사용할수 있다.

5. KDE와 Qt

KDE의 사본을 구입할 때 그에 일치하는 Qt의 판도 따라온다. KDE와 Qt는 항상 갱신되므로 호환성을 보증하기 위하여 KDE와 함께 공급된 Qt의 판을 사용하는것이 중요하다. 그 일부는 오직 안정(출하)판만 제공하고 다른 일부는 불안정한(개발)판을 제공한다. 그러므로 KDE의 출하판을 리용하겠는가 아니면 최종개발판을 리용하겠는가 결심해야 한다. 그 결심은 사용자가 무엇을 하려고 하는가에 달려있다.

안정판을 선택한다면 mirror싸이트를 하나 선택하고 README파일의 지시를 따라 자기 Linux판에 적합한 등록부들을 선택해야 한다. KDE에서 몇가지 다른 묶음선택을 사용할수 있다. Red Hat에는 RPM파일들, Debian에는 DEB파일들이 그리고 일반설치용으로 tar파일들이 있다.

6. 최근판의 구입과 설치

최신판의 쏘프트웨어를 얻어서 갱신하려고 한다면 마지막으로 내리적재한 후 변경된것들만 포함하는 갱신내용을 계속 내리적재하는 방법으로 모든 원천코드를 내리적재한다.

1) CVS쏘프트웨어

CVS(병행판체계)는 프로그람의 원천파일들에 대한 변화궤적을 보유하는 원천코드조종체계이다. KDE와 Qt는 모두 CVS에 의해 관리된다. 자기 체계에 cvs라는 편의프로그람이 있어야 한다.

쏘프트웨어를 개발하려면 CVS쏘프트웨어에 대하여 알아야 한다. CVS는 원천코드모듈을 검사하여 같은 원천코드의 여러판의 개발을 방지하는데 쓰이므로 여러명의 사람들이 쏘프트웨어를 변경하려고 할 때 특별히 필요하다.

2) Creating a CVS환경파일의 만들기

CVS를 실행하기 전에 미리 설치되여야 할것이 있다. 자기 홈등록부에서 다음 내용으로 .cvsrc라는 본문파일을 만든다.

```
cvs -z4 -q
diff -u3 -p
update -dP
checkout -P
```

선택설정에 문자 p를 쓰는데 주의하여야 한다. Diff행에서 그것은 소문자이고 update와 checkout행에서는 대문자이다.

3) CVSROOT환경변수의 설정

원격싸이트에서 원천을 발견할 장소를 CVS에게 알리는 환경변수를 설정해야 한다.

다음과 같이 입력한다.

```
export CVSROOT-=:pserver:anonymous@anoncvs.kde.org:/home/kde
```

4) 원격CVS봉사기에 가입

원천나무의 부모등록부로 하려는 등록부를 만든다. 자기가 선택한 KDE의 각 부분은 보조등록부를 만드므로 새로운 등록부로 변경하고 다음 지령을 입력한다.

```
cvs login
```

암호를 요구할수 있는데 Return 또는 Enter를 누른다. 봉사기의 응답에 시간이 걸릴수 있으므로 몇분 기다릴수 있다.

5) 원천파일사본의 내리적재

다음의 지령목록은 모든 원천을 내리적재한다. (스크립트로 편집되여 헌번에 실행될수

있다.) 모두 요구하지 않으면 이 지령들중 일부를 뺄수 있다.

cvs checkout kde-qt-addon

cvs checkout qt-copy

cvs checkout kdelibs

cvs checkout kde-i18n

cvs checkout kdeadmin

cvs checkout kdebase

cvs checkout kdegames

cvs checkout kdegraphics

cvs checkout kdemultimedia

cvs checkout kdenetwork

cvs checkout kdesdk

cvs checkout kdesupport

cvs checkout kdetoys

cvs checkout kdeutils

cvs checkout kdevelop

cvs checkout kdoc

cvs checkout kfte

cvs checkout klyx

cvs checkout kmusic

cvs checkout koffice

cvs checkout korganizer

cvs checkout ksite

cvs checkout kdepim

6) 변경의 유지

마지막 내리적재이후 변경된 원천을 내리적재할 때마다 원천코드의 새로운 사본을 구입할수 있다. 원래의 내리적재시에 수행한 모든 조작을 한다. 그러나 cvs지령행에서 검사하지 않고 update를 리용한다. 실례로 qt-copy등록부의 최신판을 내리적재하려면 다음 지령을 입력한다.

cvs update qt-copy

모든것은 최신판으로 갱신하려면 검사에서 리용한 스크립트를 리용하지만 모든 지령을 update로 바꾸어야 한다. 다시 스크립트를 실행할수 있다.

7) 코드의 콤파일

원천코드의 매개 등록부는 개별적으로 콤파일되여야 한다. 우선 콤파일해야 하는 등록부는 qt-copy등록부이다. 그것은 거의 모든 다른 등록부들이 여기에 의존하기때문이다.

qt-copy등록부를 변경하고 다음과 같이 4개 지령을 입력한다.

```
make -f Makefile.cvs
./configure -sm -gif -system-libpng -system-jpeg
make
make install
```

그다음 나머지 매개 등록부에서 다음의 4개 지령을 입력한다.

```
make -f Makefile.cvs
./configure
make
make install
```

각이한 등록부들에 의해 생성된 서고들중에 일부 의존관계가 있다. 일부 등록부는 다른 등록부들보다 먼저 콤파일되여야 한다. 우선 qt-copy을 콤파일하고 그 다음 kde-qt-addon, kdebase 그리고 kdelib를 콤파일해야 한다.

모두에 대한 삭제콤파일을 할 때에는 등록부들의 순서를 반전해야 한다. 4개 지령중 처음 2개는 한번만 입력해야 한다. 등록부를 재시동해야 한다면 3번째 지령으로 기동한다.

참고문헌

Jusmin Blanchette, Mark Summerfield 《C++ GUI Programming with Qt 3》 Prentice Hall, 2004.

이 책은 콤퓨터를 전공으로 하는 교원, 연구사, 대학생들을 위한 참고서이다.

KDE/Qt프로그람개발법

집 필	한영철	**심 사**	박사 정강호, 김순옥 박사 부교수 김기준, 김영일
편 집	차현옥	**교 정**	서금석
장 정	서경애	**콤퓨터편성**	여은정
낸 곳	교육위원회 교육정보쎈터	**인쇄소**	
인 쇄		**발 행**	

교-09-1820　　부　　값　　원